Journal of Korea Multimedia Society Vol. 10, No. 11, November 2007(pp. 1460-1471)

자바 기반 휴대용 임베디드 기기의 삼차원 엔진 성능 향상을 위한 바인딩 구현

김영옥[†], 노영섭[††]

요 약

휴대용 임베디드 기기에서의 삼차원 엔진은 크게 바이트 코드를 실시간으로 해석하여 실행하는 자바 기반의 JSR184와 C언어 기반의 OpenGL/ES가 있다. 이들 두 표준에서 자바 객체를 지원하는 JSR184는 OpenGL/ES에 비하여 상대적으로 많은 프로세서의 자원을 사용하여 제한된 연산능력을 보유하고 있는 임베디드 기기에 적용할 경우 제약이 따를 수 밖에 없다. 반면에 기존 개인용 컴퓨팅 환경에서 사용되는 삼차원 컨텐츠는 자바의 장점을 이용하여 제작되었기 때문에 유럽에서 많은 사용자 층을 확보하고 있고, 또한 그 컨텐츠의 품질이 우수하여 상용 통신망인 GSM 망에서 많이 서비스 되고 있다. 따라서 GSM 망에서 사용되는 휴대용 임베디드 기기에 기존의 자바 기반 삼차원 컨텐츠를 별도의 변환 과정 없이 지원할 수 있는 JSR184의 지원이 필요하지만, 현재 개발되어 사용되는 자바 기반 삼차원 엔진은 휴대용 기기가 보유한 연산능력에 비하여 상대적으로 많은 연산량을 필요로 하기 때문에 상용제품에 적용하기에 많은 어려움이 따른다. 본 논문에서는 휴대용 임베디드 기기가 가지고 있는 충분하지 않은 연산능력을 바탕으로 자바 객체의 장점을 수용하면서 삼차원 컨텐츠의 처리속도를 향상 시킬 수 있는 바인딩 기법을 제안하였다. 제안된 바인딩 기법은 자바를 이용한 삼차원 컨텐츠를 지원하기 위하여, JSR184의 표준 인터페이스를 상위 계층에서 지원하고, OpenGL/ES와 JSR184를 서로 연결하기 위하여 이기종 코드 변환 언어인 KNI(Kilo Native Interface)를 중간 계층에서 사용하였고, 하위 계층에서 OpenGL/ES의 표준을 구현하였다. 제안하는 바인딩 기법은 모의 실험을 통하여 기능을 검증하였고, ARM을 장착한 FPGA를 사용하여 그 성능을 평가하였다.

Design of a Binding for the Performance Improvement of 3D Engine based on the Embedded Mobile Java Environment

Young-Ouk Kim[†], Young-Sup Roh[††]

ABSTRACT

A 3-Dimensional engine in a mobile embedded device is divided into a C-based OpenGL/ES and a Java-based JSR184 which interprets and executes a byte code in a real-time. In these two standards, the JSR184 supporting Java objects uses more processor resources than an OpenGL/ES and thus has a constraint when it is used in an embedded device with a limited computing power. On the other hand, 3-Dimensional contents employed in existing personal computer are created by utilizing advantages of Java and secured numerous users in European market, due to the good quality in contents and extensive service in a commercial network, GSM. Because of the reason, a mobile embedded device used in a GSM network needs a JSR184 which can provide an existing Java-based 3-Dimensional contents without extra conversion processes, but the current version of Java-based 3-Dimensional engine has drawbacks in application to commercial products because it requires more computing power than the mobile embedded device. This paper proposes a binding technique with the advantages of Java objects to improve a processing speed of 3-Dimensional contents in limited resources of a mobile embedded device. The technique supports a JSR184 standard interface in the upper layer to utilize 3-Dimensional contents using Java, employs a different code-conversion language, KNI (Kilo Native Interface), in the middle layer to interface between OpenGL/ES and JSR184, and embodies an OpenGL/ES standard in the lower layer. The validity of the binding technique is demonstrated through a simulator and a FPGA embedding an ARM.

Key words: Java(자바), Kilo Native Interface(KNI, 이기종 변환 언어), OpenGL/ES(오픈지엘/이에스), JSR184(제이에스알184), 3D Graphics(삼차원 그래픽스), Mobile Embedded Device(휴대용 임베디드 기기)

1. 서 론

휴대용 임베디드 기기의 삼차원 엔진은 크게 자바 언어로 표준이 정립된 JCP(Java Community Process) 그룹의 JSR184[1]와 C 언어로 표준이 정립된 크로노스 그룹의 OpenGL/ES[2]가 있다. JSR184는 언어가 가지고 있는 자바 객체의 특성을 활용하고 기존 Java3D 클래스가 안고 있는 무거움을 줄였고, 휴대용 기기의 성능을 고려하여 초경량으로 구현할 수 있도록 기능이 정의되어 있어[1], 구현된 삼차원 엔진이 소형이며 적은 메모리를 사용한다. 그러나 이런 장점에도 불구하고 제한된 연산 능력을 보유한 휴대용 기기의 플랫폼에서 JSR184를 구동하면 실행 코드가 바이트 형태로 변경되어 수행되므로 처리 속도가 늦다[1,3].

이와 대조적인 크로노스 그룹의 OpenGL/ES는 C 형태의 저수준에서 하드웨어를 이용한 렌더링 방식을 채택하였다. 이 방법은 자바에 비해 삼차원 그래픽의 처리 속도가 빠르나 기존의 자바 컨텐츠를 수용하지 못하고 JSR184에서 지원하는 애니메이션이나 고수준의 기하 처리 연산을 지원하지 못하는 단점이 있다[2]. 그리고 처리속도의 향상을 위해 객체화된 컨텐츠 대신 프로시져 형식의 컨텐츠를 이용하기 때문에 두 그룹 간 표준에서 차이가 발생하여 컨텐츠의 재사용이 어렵고, 서비스를 위해 중복된 작업을 유발한다.

또한 JSR184의 표준 제정 당시 OpenGL/ES 표준은 초기 버전이 제안된 상태였으므로 JCP 그룹은 속도 보다 삼차원에 사용되는 각종 기법들 (예를 들어 몰핑, 애니메이션, 키 프레임 등) 위주로 JSR184의 표준을 제정하였기 때문에 나중에 제정된 자바 기반의 JSR184가 수행성능이 떨어지게 되는 결과를 낳았다.

그러나 GSM 계열의 휴대용 임베디드 기기가 사용되는 시장 환경과 OSMU(One Source Multi User) 환경에서 기존에 개발된 자바 기반 삼차원 컨텐츠를 휴대용 임베디드 기기에서 변환 없이 사용하고자 하는 요구가 많이 있어 JSR184의 지원이 필요하지만, 자바 가상 기계에서 구동되는 삼차원 컨텐츠의 기하 처리와 부동 처리의 과도한 연산은 JSR184의 삼차원 컨텐츠 구동에 부하로 작용하여, 제한된 연산능력을 보유한 휴대용 임베디드 기기에서의 자바 기반 삼차원 컨텐츠 사용이 원활하지 못하게 된다. 따라서 본 논문에서는 휴대용 임베디드 기기에서 기존 자바 컨텐츠의 사용이 가능하도록 JSR184를 OpenGL/ES와 연결하여 처리 성능을 향상 시키는 방법을 구현하고 성능을 측정하였다. 이 방법은 JSR184 이후 JCP 그룹이 JSR239를 통해 해결하고자 관련 표준을 발표하였으나 단지 영상의 출력에 해당되는 EGL(LCD 출력)과 기타 API 선언만 있을 뿐 내부적의 구현에 대한 언급은 없다[4]. 본 논문은 JSR184와 OpenGL/ES의 내부 파이프라인을 비교한 후 동일 파이프라인 및 연산 오버로드 부분을 파악하여 이기종 변환 언어인 KNI로 연결 하고 과도한 기하 연산 처리는 저수준의 C 형태로 변환하여 성능을 향상 하였다. 이후 관련 어플리케이션이 구동되는 상위 유저 인터페이스는 자바를 지원하여 삼차원과 이차원 표현이 가능하게 하고 물체를 표현하는 렌더링 가속 및 전체 속도 향상을 위해 JSR184 내부를 개선하여 OpenGL/ES로 바인딩 하는 인터페이스를 통해 자바 기반 삼차원 컨텐츠의 원활한 구동이 가능하게 하였다[1,2].

2. 관련 연구

휴대용 삼차원 엔진의 API는 실시간으로 사용자의 입력을 반영해 컨텐츠를 구동하고, 영상 데이터를 변형, 가공하여 장면을 생성해 내는 기하학 연산 처리기와 장면 데이터의 색상 및 텍스쳐, 광원, 안개등의 효과를 처리하는 렌더링 처리기로 이루어진다.

일반적인 PC 기반의 고성능 컴퓨팅 환경에서는 위의 전.후처리를 그래픽 카드의 하드웨어 부분이 수행하여 빠른 처리가 가능 하나 모바일 환경에서는 자원의 한계를 극복하기 위하여 주로 렌더링 부분을 별도의 하드웨어로 구현하여 사용한다. 따라서 기하학 연산 처리기는 많은 부분을 소프트웨어로 처리하

※ 교신저자(Corresponding Author) : 노영섭, 주소 : 서울시 강남구 삼성동 37-18(135-867), 전화 : 02)3470-5282, FAX : 02)3470-5282, E-mail : ysroh@suv.ac.kr
접수일 : 2007년 4월 10일, 완료일 : 2007년 10월 2일
† 정회원, 서울벤처정보대학원대학교 임베디드시스템학과 (E-mail : kirra@corelogic.co.kr)
†† 정회원, 서울벤처정보대학원대학교 임베디드시스템학과 교수
※ 이 연구는 서울시 산학연 협력 사업 지원으로 연구되었음

게 되고 수행 결과를 하드웨어로 넘겨주는 방식을 사용하는 것이다. 현재 표준으로 제정된 휴대용 삼차원 엔진인 JSR184는 장면 생성을 위해 각 오브젝을 관리하고 각종 기하 연산 및 애니메이션을 수행하는 기하학 연산 처리기에 초점을 맞추어 설계 되었다. 반면 OpenGL/ES는 저수준의 하드웨어 렌더링에 근접한 빠른 속도와 단순한 파이프라인을 갖고 있어 CPU 자원의 소모를 최대한 줄여 성능을 향상 했다.

2.1 자바 표준의 휴대용 삼차원 그래픽 API

JSR184는 고수준 API를 지원하는 자바 기반 오브젝 형태의 표준 삼차원 그래픽 API이다. JSR184는 삼차원 엔진을 수행하기 위해 약 30개의 클래스로 이루어져 있으며 그림 1과 같이 Graphics3D, World, Group, 그리고 Mesh 객체를 통해 삼차원 자원을 관리하고 장면을 형성한다. 그림 1은 JSR184의 계층 구조도를 나타낸다.

여기서 World 객체는 장면 그래프의 루트 노드로서 여러 객체의 최상위 층에 해당되며 World 객체를 통해 렌더링에 필요한 모든 객체를 연결한다. 또한 Graphics3D 객체는 자바의 다른 객체의 결과와 연동 및 최종 결과물을 저장할 프레임 버퍼와 연결할 수 있으며 LCD등의 정의된 출력 장치를 담당한다.

JSR184는 삼차원 장면을 처리하기 위한 기하학적 연산 처리 부분을 중점적으로 명시하였으며, 실제 삼차원을 픽셀로 변환하여 프레임 버퍼에 장면을 생성하는 부분은 특별히 명시하지 않았다. 따라서 이 부분은 하드웨어와 밀접한 관계를 갖는다[5].

JSR184와 OpenGL/ES의 이기종간 인터페이스 연결을 위해 애니메이션 제어, 키 이벤트, 및 관련 어플리케이션이 구동되는 최상위 계층은 자바 형태로 남겨두고 속도 향상을 위해 OpenGL/ES와 바인딩 되는 부분을 각각의 객체별로 구별하여 클래스를 선별하여야 한다.

그림 1에서 World, Group, Background, User-Object, Mesh, Camera, 그리고 Light 클래스들은 JSR184의 계층 중 삼차원 데이터 및 장면을 구성하는데 있어 중요한 클래스들이다[1]. 여기서 OpenGL/ES API와 바인딩 할 수 있는 클래스를 살펴보면 기하학적 연산 처리를 구성할 수 있는 데이터 입력인 UserObject, Group, Mesh, Camera, 그리고 Light 클래스가 있으며, 후처리 부분인 렌더링은 위의 처리 결과를 이용해 Graphics3D와 연결하면 된다[2,5].

2.2 저수준의 내장형 삼차원 그래픽 API

OpenGL/ES는 크로노스 그룹이 제안한 저수준 그래픽 라이브러리 API이다. 이 API는 휴대용 기기에서 삼차원 렌더링 엔진을 수행하기 위해 제안된 표준이며, 작은 크기의 기하학 요소를 갖는 포인트, 라인, 폴리곤, 이미지, 그리고 비트맵 등을 처리하는 부분과 정점 연산, 기초 연산, 그리고 픽셀 연산 등을 처리하는 전처리 부분 및 단위 연산, 텍스쳐 연산, 그리고 전체 프레임 버퍼를 처리하는 후처리 부분으로 이루어져 있다. OpenGL/ES의 렌더링 프로세스는 그림 2와 같다.

그림 2에서 정점 연산은 삼차원을 구성하기 위한 기본 요소인 정점들을 버텍스 데이터로부터 받아 4 × 4 부동 소수점 행렬 형태로 구성하는 역할을 한다. 이것은 잘라내기 및 삼차원 물체를 구성하기 위한

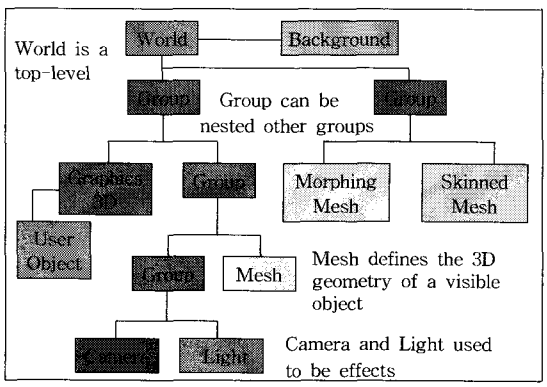


그림 1. JSR184 계층 구조도

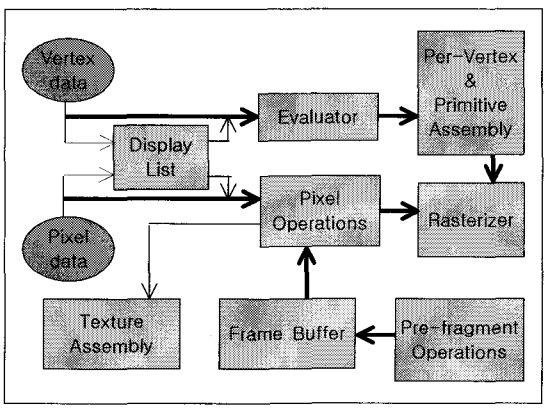


그림 2. OpenGL/ES 렌더링 파이프라인 프로세스

삼각형 셋업 기초 연산을 통해 앞으로 보여질 공간상에 위치를 잡는다.

픽셀 연산은 위의 정점 데이터가 OpenGL/ES 렌더링 파이프라인으로 처리되는 반면, 여러 경로를 통해 다르게 처리된다. 먼저 시스템 메모리로부터 다양한 포맷으로 저장되어 있는 픽셀들을 풀어 다음 연산 단계의 각 블록으로 집어넣는다. 그리고 이러한 데이터들에 대해 스케일 바이어스 연산을 수행하고 픽셀 맵에 따라 처리하여 그 결과를 클램프 시키고, 이를 텍스쳐 메모리에 쓰거나 라스터화 단계로 전송하는 역할을 한다.

텍스쳐 연산은 현실감 있는 표현을 위해 삼차원으로 구성된 여러 오브젝에 텍스쳐 이미지를 붙이는 역할을 한다.

OpenGL/ES의 최종 단계로 라스터라이저는 기하 데이터와 픽셀 데이터 모두 프레그먼트들로 변환을 하여 프레임 버퍼에 픽셀을 채우고 이때 선과 폴리곤 스타일, 라인 폭, 점의 크기, 쉐이딩 모델, 그리고 안티알리어싱 등을 지원하기 위해 적용 범위등을 고려하여 최종 영상을 만들어 낸다. OpenGL/ES의 전체 블록 기능은 그림 3과 같다.

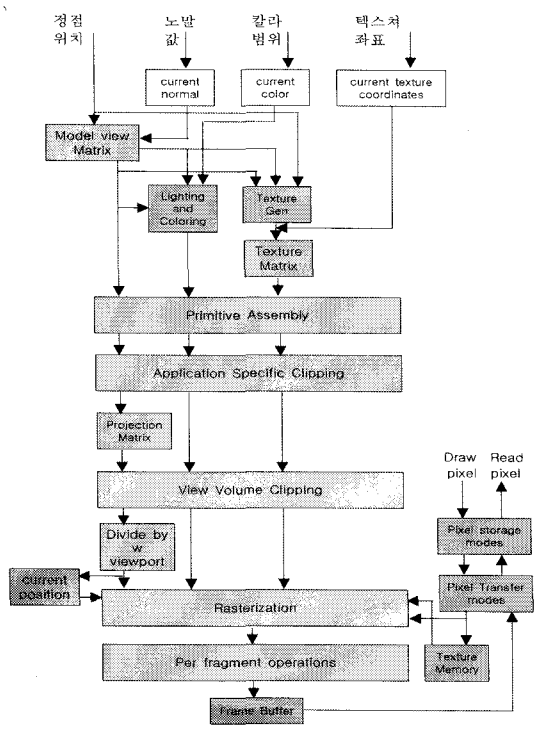


그림 3. OpenGL/ES 기능 블록 다이어그램

2.3 JSR184와 OpenGL/ES의 바인딩 기법

본 논문에서 제안하는 바인딩 기법은 자바의 장점을 살림과 동시에 저수준의 빠른 렌더링 처리 기법을 통해 객체의 장점을 수용하고 하드웨어에 근접한 속도 향상을 내기 위한 방법으로 구현되었다. 먼저 JSR184가 가지는 장점을 최대한 수용하기 위해 객체로 삼차원 장면을 생성하는 주요 클래스인 Graphics3D를 중점적으로 World, Mesh, 그리고 Group 등의 클래스를 OpenGL/ES의 장면을 생성하는 API와 관계를 따져 이기종 변환 언어로 변환할지의 여부를 결정한다. 제안하는 기법으로 렌더링 처리 속도 향상을 위해 다음과 같은 형태로 바인딩 할 파이프라인의 관계도를 정형화한다. 그림 4의 JSR184 부분은 자바의 객체를 수용하기 위한 상위 계층이고 OpenGL/ES와의 공통 파이프라인은 바인딩 기법을 통해 처리 속도를 향상할 부분이다.

3. JSR184와 OpenGL/ES를 연결할 바인딩의 구현

자바 객체로 정의된 JSR184는 객체를 정의하고 설계할 때, DSP(Digital Signal Process)형태의 구조를 바탕으로 설계되었기 때문에 JSR184 자체를 하드웨어로 구현하기 쉽다[1]. 이것은 JSR184의 외부 구조는 객체의 장점을 수용하지만, 내부 객체는 삼차원 엔진을 설계하는 일반적인 파이프라인 구조를 이용했음을 보여준다. 또한 OpenGL/ES API도 모바일 환경에서 하드웨어 가속도를 사용 할 수 있는 표준이므로 이기종간 바인딩 인터페이스 범위의 확정은 두 API간의 파이프라인 흐름을 보며 결정할 수 있다.

2.1절에서 언급한 바와 같이 기하학적 연산 처리를 담당하는 클래스를 일차적으로 선별하고 DSP 성격을 갖는 클래스를 이차 요소로 선택 범위를 축소하였다. 다만 클래스의 선택이 잘못되면 부가적인 코드가 더 소요되어 CPU 자원이 낭비되므로 주의를 요해야 한다.

위의 두 요소와 각각의 클래스 관계 및 부합되는 구조를 충분히 분석한 후 정점 버퍼, 광원 및 물질, 그리고 카메라 클래스를 선별 하였고, 기타 기하 연산 처리를 담당하는 클래스 중 일부 함수를 사용하였다. 또한 본 논문에 구현된 샘플 코드의 각 부분에 바인딩 된 부분을 주석으로 표시 하였다.

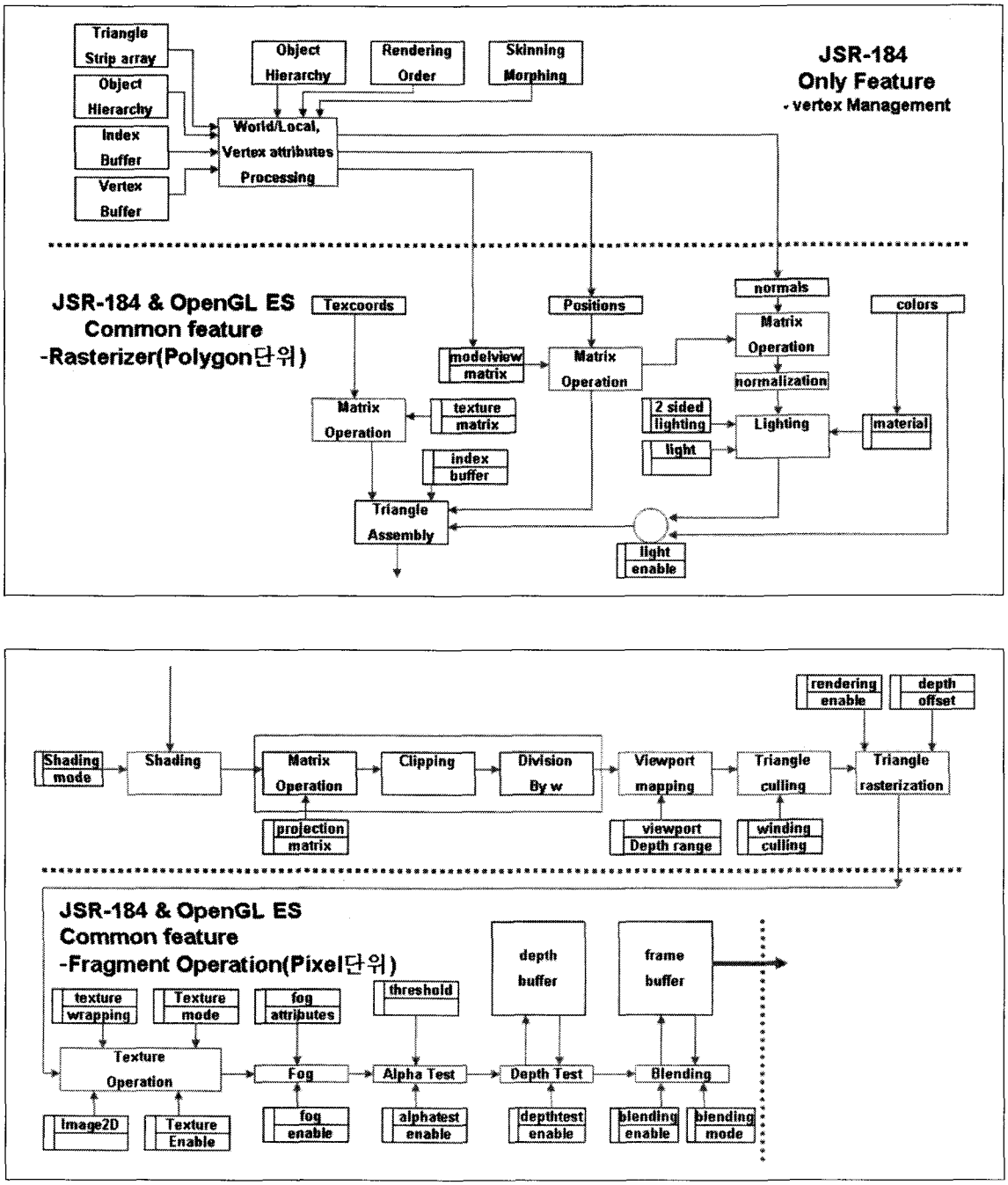


그림 4. 제안하는 JSR184와 OpenGL/ES의 바인딩 파이프라인 매치도

1. 버텍스 버퍼 클래스

버텍스 버퍼 클래스는 삼차원을 구성하는 세점 x,y,z 좌표와 텍스쳐 위치, 법선(Normal), 그리고 칼라 좌표의 값을 입력 받아 물체의 형태 및 색상 등을 처리한다. 따라서 이 클래스는 OpenGL/ES의 정점, 칼라, 법선, 그리고 텍스쳐 위치 값을 처리하는 API, 즉 glVertexPointer, glNormalPointer, glColor-Pointer, 그리고 glTexCoordiPointer 함수와 바인딩

을 시킨다.

다음은 버텍스 버퍼 클래스를 바인딩 하기 위한 자바 클래스 구현의 원시 소스와 KNI로 변형한 이기종 C 소스의 일부분이다.

VertexBuffer.java

```
public class VertexBuffer extends Object3D {
  public VertexBuffer()
  {
      this(createNativeVertexBuffer(),
          VERTEXBUFFER);
  }
    private setPositions(VertexArray obj, float
          scale, float[]attr)
  {
    setNativePositions(obj, scale, attr);
  }
  private setNormals(VertexArray obj)
  {
    setNativeNormals(obj);
  }
  private setColors(VertexArray obj)
  {
    setNativeColors(obj);
  }
    private setTexCoords(int idx, VertexArray
          obj, float scale, float[] scalebias)
  {
    setNativeTexCoords(idx, obj, scale,
          scalebias);
  }

  /*Declaration of the Natives Functions For
Binding API */
    private native void
          setNativePositions(VertexArray obj,
          float scale, float[] attr);
    private native void
          setNativeNormals(VertexArray obj);
    private native void
          setNativeColors(VertexArray obj);
    private native void setNativeTexCoords (int
          idx, VertexArray obj, float scale,
          float[]scalebias);
}
```

Native_kni_vertexbuffer.c

```
KNIEXPORT KNI_RETURNTYPE_VOID
  Java_javax_microedition_m3g_VertexBuffer_se
  tNativePosition()
{
  /* 중략 */
  /* Binding for Vertex Process */
  glVertexPointer(pstVertexPos->nNoComponen
          ts, GL_FLOAT, 0,pstPosition-
          Vertex->pfGLVertex);
}
```

위의 버텍스 버퍼 클래스는 순수 JSR184로 구현된 자바 함수를 KNI 형태로 재 선언하고 바인딩 할 내부 함수는 KNI 형태로 변경한 것이다. KNI 인터페이스와 연결될 바인딩 함수는 C 형태로 구현하여 Open GL/ES API인 정점 연산의 주요 함수와 연결했다.

3.2 광원 및 물질 클래스

광원 클래스는 삼차원을 구성하는 임의의 물체에 빛의 효과를 주는 역할을 하며 상기 클래스는 OpenGL/ES의 glLight와 glMaterial 계열의 함수와 바인딩이 가능하다. 단, JSR184 표준에 명시된 라이트 관련 표준과 OpenGL/ES간 라이트 표준의 상이함 때문에 공통된 부분만 찾아 바인딩 시키기로 한다. 표 1은 OpenGL 버전 1.3의 물질 표준이며 공통된 부분은 표 1을 따른다.

표 1에서 표준의 권고사항은 OpenGL 버전 1.3의 라이팅 방정식[6]에 명시된 사항이며 JSR184는 위의 표준을 따른다. 따라서 OpenGL/ES와의 라이팅 관련 바인딩 함수는 제한적으로 진행되었으며 구현시 일부 함수의 파라미터만 사용하였다. 아래는 바인딩 된 라이트 함수의 내용이며 접두사 gl 로 시작하는 함수가 OpenGL/ES와 연결된 부분이다.

표 1. OpenGL 버전 1.3의 물질 표준

표준 권고사항	해당 사항
Secondary color	not supported
Front/Back face	the same Material used for both
Color Tracking	limited to AMBIENT_AND_DIFFUSE
Intensity	ambient, diffuse and specular are 0
Global Ambient	ambient color alpha is not supported

```
void applyLight(udtLight *pstLight, int LightID)
{
  /* Binding for Light and Material */
  glLightModelfv(GL_LIGHT_MODEL_AMBIEN
                T, LIGHT_ZERO);
  if(pstLight->nMode == AMBIENT)  //if mode
   is Ambient
  {
    glLightfv(LightID, GL_AMBIENT, fColor);
    glLightfv(LightID, GL_DIFFUSE,
              LIGHT_ZERO);
    glLightfv(LightID, GL_SPECULAR,
              LIGHT_ZERO);
  } else    //else if mode is Directional/Omni/Spot
  {
    /* Binding for Lighting Element */
    glLightfv(LightID, GL_AMBIENT,
              LIGHT_ZERO);
    glLightfv(LightID, GL_DIFFUSE, fColor);
    glLightfv(LightID, GL_SPECULAR, fColor);
  }
  /* Binding for GL_POSITION */
  glLightfv(LightID, GL_POSITION,
       pstLight->nMode != DIRECTIONAL ?
       LOCAL_ORIGIN : POSITIVE_Z_AXIS);
  glLightfv(LightID, GL_SPOT_DIRECTION,
       NEGATIVE_Z_AXIS);

  if(pstLight->nMode == SPOT)
    {
    /* Binding for Spot Lighting */
    glLightf(LightID,GL_SPOT_EXPONENT,
             pstLight->fSpotExponent);
    glLightf(LightID, GL_SPOT_CUTOFF,
             pstLight->fSpotAngle);
  } else    //else if ambient/directional/omni
  {
    glLightf(LightID, GL_SPOT_CUTOFF,
             180.0f);
    }

  if(pstLight->nMode == OMNI ||
    pstLight->nMode == SPOT )
    {
    /* Binding for Attenuation */
    glLightf(LightID,
             GL_CONSTANT_ATTENUATION,
             pstLight->fAttenuationConstant);
    glLightf(LightID,
             GL_LINEAR_ATTENUATION,
             pstLight->fAttenuationLinear);
    glLightf(LightID,
             GL_QUADRATIC_ATTENUATION,
             pstLight->fAttenuationQuadratic);
  } else if(pstLight->nMode == AMBIENT)
  {
    /* Binding for AMBIENT Lighting */
    glLightf(LightID, GL_CONSTANT_
             ATTENUATION, 1.0f);
    glLightf(LightID, GL_LINEAR_
             ATTENUATION, 0.0f);
    glLightf(LightID, GL_QUADRATIC_
             ATTENUATION, 0.0f);
  }
}
```

3.3 카메라 클래스

카메라 클래스는 삼차원 물체를 공간상에 위치할 때 물체가 놓이는 좌표를 연산한다. 카메라는 두 가지 모드에 의해 물체를 바라보는 시각이 다르다. 일반적으로 흔히 사용되는 직교 모드와 원근감 형태가 있으며 장면을 구성하는 모든 물체에 4×4 형태의 매트릭스 값으로 곱해져야 하기 때문에 빈번이 사용되고 파이프라인에서 중요한 기능을 한다. 따라서 기하 연산 처리의 연산 시간을 줄이기 위해 OpenGL/ES의 기하 부분과 필수적으로 바인딩이 필요한 부분이다.

다만 OpenGL/ES의 논리적 좌표는 좌측 하단이 (0,0)으로 중점을 잡는 반면 모바일 휴대용 기기의 LCD는 좌측 상단을 중점으로 사용한다. 따라서 원근감 매트릭스 변환을 할 때 동차 좌표를 사용하여 물체를 논리적 공간에 위치시키고 시점 좌표를 연산하는 마지막 시점에서 물체를 이차원 공간상에 위치시킬 때 y 축의 부호를 바꾸어 전체 장면을 뒤집어 출력한다.

물체의 잘라내기 함수 바인딩은 법선 좌표 장치(NDC) 이전의 값을 이용하여 정규화 해야 한다. 만약 법선 좌표 장치 이후의 값으로 바인딩 되면 안개 효과 및 기타 랜더링 처리시에 연산되는 각종 효과에 영향을 미칠 수 있으므로 주의한다. 다음 예제 코드는 카메라 클래스에서 원근감을 구현한 바인딩 함수의 일부이다.

```
void setCameraPerspective(udtCamera
     *pstCamera, float fFovy, float fAspectRatio,
     float fNear, flaot fFar)
{
  if(fFovy <= 0 || fAspectRatio <= 0 || fNear <=
    0 || fFar <= 0 || fFovy >= 180)
    KNI_ThrowNew("java/lang/IllegalArgument
    Exception", "Perspective error");
  else
  {
    pstCamera->fFovy = fFovy;
    pstCamera->fNear = fNear;
    pstCamera->fFar = fFar;
    pstCamera->fAspectRatio = fAspectRatio;
    pstCamera->nProjectionType =
                  PERSPECTIVE;
    fHeight= (float)tan(fFovy *PI / 360.0);
    fWidth = fHeight * fAspectRatio;
    fDepth = fFar - fNear;

    if(fDepth)
    {
      /* Binding for Perspective Matrix */
      Set_Perspective_Matrix(pstCamera->
          arrfMatrix,pstCamera);
    } else
    {
      /*view volume is zero */
      memset(pstCamera->arrfMatrix,0,sizeof
            (float)*16);
    }

    /* Binding for Camera and Matrix */
    transposeMatrix(pstCamera->affMatrix,
                  affMatrix);
    glMatrixMode(GL_PROJECTION);
    glLoadMatrix(affMatrix);
    glMatrixMode(GL_MODELVIEW);
  }
}
```

3.4 기타 기하 관련 클래스

Graphics3D에서 장면의 크기 및 합성 클래스에서 사용할 깊이 오프셋, 깊이 범위, 컬링, 영역지정, 라인 크기, 그리고 포인트 사이즈 등의 기본 속성에 관련된 기능을 OpenGL/ES와 바인딩 한다.

4. 실험 및 검토

본 논문에서 구현한 OpenGL/ES 바인딩은 휴대용 임베디드 기기에서 자바 삼차원 엔진의 성능향상을 목적으로 설계하였으나 실제 삼차원을 픽셀로 변환하여 LCD 등의 출력 장치에 출력하는 렌더링 부분은 하드웨어와 연결이 가능하며 이 부분은 구현하는 방법마다 성능의 차이가 다르다[5]. 따라서 하위 부분과 연결되는 드라이버 및 실제 칩 상에서의 속도 측정은 실험에서 제외되었으나 FPGA를 이용한 ARM 버전의 이진 코드와 이를 수행 시키고 분석하는 ARM Profiler를 이용하여 최종 렌더링 엔진의 속도를 측정하였다. 또한 이와 동일한 인터페이스를 보장하고[2] 윈도우즈에서 구동되는 OpenGL/ES를 이용하여[7] PC 기반에 구현된 자바 엔진과 연결하여 실험하였다[8].

성능향상 측정을 위한 구조는 바인딩 인터페이스를 자바 버추얼 머신(JVM)의 이기종 코드 변환 언어인 KNI(Kilo Native Interface) 형태로 코드를 작성하고 외부 자바 클래스와 내부 코드를 OpenGL/ES로 연결하여 에뮬레이터[9]로 모의실험을 하였다.

사용한 에뮬레이터는 JVM을 윈도우에서 구동시키며, 휴대용 기기에서 J2ME를 수행시키기 위한 통합 플랫폼을 제공한다. 바인딩 인터페이스 시험을 위해 자바로 구현한 순수 JSR184 클래스와 KNI로 변경하여 이기종간 연결한 클래스 각각을 확인하여 윈도우즈에서 실험하였다.

JSR184 표준에 명시된 각각의 클래스의 이기종간 연결하는 인터페이스 설계는 다음 순서에 따랐다. 먼저 해당 클래스의 주요 함수를 JVM에서 KNI 형태로

등록하기 위해 'OemNativesTables' 형태로 컴파일을 하였다. 컴파일된 해당 클래스의 함수는 KNI 인터페이스 형태로 정의하고 재 컴파일하면 JVM이 해당 함수를 내부 테이블에 등록한다. 이후 객체에서 관련 함수가 불려 지게 될 때, JVM이 테이블을 참조하여 해당 바인딩 함수를 실행 시킨다[3]. 아래 키워드 'NativeFunction'은 성공적으로 컴파일 완료시 JVM 내부에 생성되는 KNI 함수 테이블이고 키워드인 'NativesTable'은 JVM이 이를 인식할 수 있는 KNI 바인딩 테이블이다.

```
static const NativeFunction
javax_microedition_m3g_VertexBuffer_natives[] =
{
  NATIVE("setdefaultColor", "(I)V",
          java_javax_microedition_m3g_Vertex
          Buffer_setDefaultColor(),
  NATIVE("setNativeColors",
          "(Ljavax/microedition/m3g/VertexAr
          ray;)V", java_javax_microedition_
          m3g_VertexBuffer_setNativeColors(),

  /* 중략 */

  {(char*)0, (char*)0, (void*)0}
};
const NativesTable natives_table[] = {

TABLE("javax/microedition/m3g/VertexBuffer",
        javax_microedition_m3g_VertexBuffer_
        natives, (NativeFunction*)0),

  /* 중략 */

  TABLE((char*)0, (NativeFunction*)0,
(NativeFunction*)0)
};
```

이와 같이 자바 버추얼 머신의 내부에 KNI 형태로 컴파일 된 코드의 장점으로 외부 인터페이스 객체는 자바의 형태를 띠지만 내부 수행은 C 형태로 동작하므로 바이트 형태의 수행속도보다 이론상 약 6배의 성능향상을 얻을 수 있다[1,2,5]. 따라서 바인딩 가능한 모든 클래스를 명시적으로 변환하여 JVM이 삼차원 엔진을 수행 시킬 때 미리 정의된 테이블에서 바로 참조 가능하게 한다.

표 2와 그림 5에서 볼 수 있듯이 실험을 위하여 삼차원을 표현하기 위한 입력 폴리곤은 1,200개, 이를 구성하기 위한 버텍스 값은 602개를 사용하여 물체를 구성하였다. 구성된 물체를 가지고 실시한 실험 결과는 순수 JSR184에 비하여 바인딩을 적용한 경우가 약 66%의 성능이 향상 되는 것으로 나타났다. 즉, 실험에서 처리된 폴리곤 수는 순수 JSR184가 초당 173,640개이고, 바인딩의 경우는 288,720개로 그래픽 처리에서 중요한 폴리곤의 처리 속도가 바인딩을 적용한 경우에 개선된 것을 알 수 있다.

다만 실험 결과로 얻어진 성능 개선이 이론적인 성능 향상 결과와 차이 나는 이유는 사용하는 자바 버추얼 머신의 성능이 다른 것이 가장 큰 이유이며, 내부적으로 바인딩은 OpenGL/ES를 별도로 구동하였기에 순수 자바 명령어를 패치 하여 구동되는 시간은 줄어들어도 OpenGL/ES의 구동시간은 영향을 받지 않기 때문이다.

표 2. 삼차원 입력 계수 및 프레임 처리율

	바인딩 유무	폴리곤 수	버택스 수	프레임/초
a	Yes	1200	602	240.6 프레임
b	No	1200	602	144.7 프레임

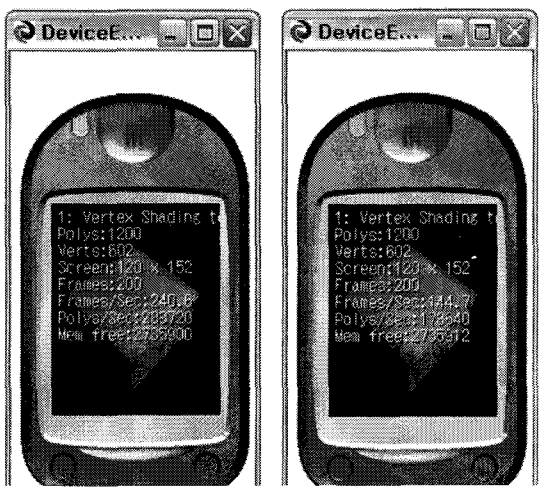
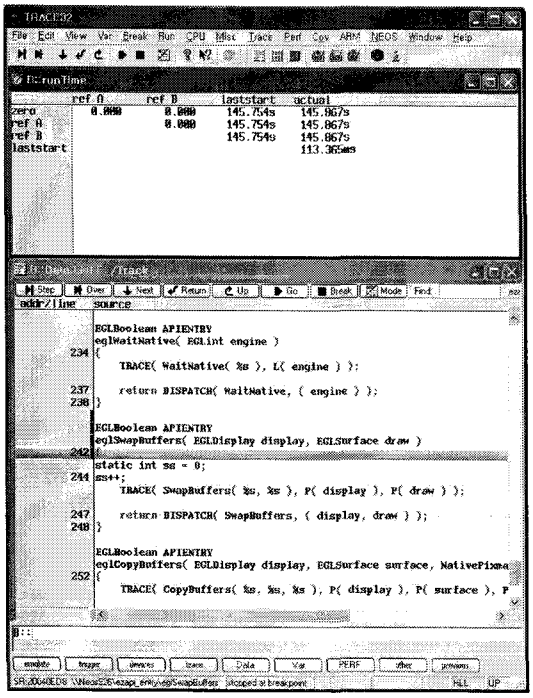
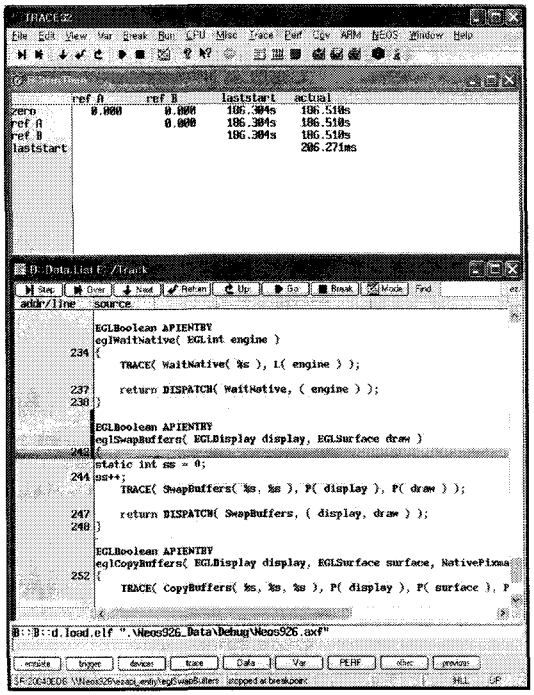


그림 5. 에뮬레이터 시뮬레이션 결과



(a) 바인딩



(b) 순수 JSR184

그림 6. ARM 프로파일러 렌더링 시뮬레이션 결과

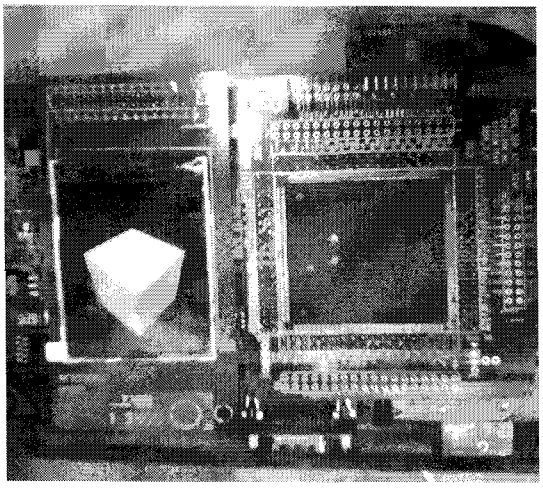


그림 7. 실제 ARM에서 구동한 바인딩 JSR184 출력

또한 휴대용 임베디드 기기의 성능 측정을 위한 다른 방법으로는 ARM 버전의 프로파일러를 사용하는 방법이 있다. 이것은 FPGA에서 이진 코드를 구동하여 시뮬레이션 시켜 수행 시간을 간접적으로 측정하는 방법이다. 실험에 사용된 방법은 바인딩이 이뤄지고 난 후 최종 출력용 픽셀 버퍼에 해당 픽셀을 채우는 렌더링 엔진의 속도를 ARM 프로파일러를 사용해 측정한 결과이다. 그림 6은 그림 5의 실험에서 사용된 동일 컨텐츠를 사용하여 얻은 프로파일링 결과로, (a)의 바인딩을 적용한 경우 한 프레임에 약 113ms의 시간이 소요되어 초당 8.82 프레임 정도가 처리되었고, (b)의 순수 JSR184의 경우는 한 프레임에 약 206ms로 초당 4.85 프레임이 처리되어, 결국 (a)의 바인딩을 적용한 경우가 (b)의 바인딩을 적용하지 않은 경우에 대하여 약 82%의 성능 개선이 있음을 확인할 수 있었다. 그림 7은 그림 6의 실험에 사용된 FPGA와 (주)코아로직의 HERA3D을 이용한 ARM 보드이며, 보드에서 사용된 ARM 코어는 926EJ-S이고 동작 주파수는 98MHz이다.

그리고 JCP 그룹이 제안한 JSR239와 비교를 통한 실험이 이뤄진다면 성능에 대한 객관적 평가를 할 수 있겠으나, JSR239의 실제 적용 사례는 각각 사용된 플랫폼, CPU, 그리고 메모리 대역폭등이 서로 달라 비교 평가가 불가능하여 실험에서 제외하였다[10].

5. 결 론

제한된 사용 환경으로 인하여 낮은 연산 능력을 보유한 휴대용 임베디드 기기에서 높은 품질을 보장하며 빠른 수행 속도를 가능하게 하는 삼차원 엔진의 설계는 PC를 기반으로 하는 엔진 설계와 많은 차이를 보인다[5,11,12]. 이는 휴대용 임베디드 기기의 낮은 연산능력을 바탕으로 고급화된 삼차원 컨텐츠를 수용하기 위해서는 일반적인 설계 방법으로는 사용자의 삼차원 컨텐츠에 대한 욕구와 휴대용 임베디드 기기의 연산능력 사이의 상반된 문제를 해결할 수 없음을 말해 준다. 따라서 본 논문에서는 휴대용 임베디드 기기가 가지고 있는 충분하지 않은 연산능력을 바탕으로 자바 객체의 장점을 수용하면서 삼차원 컨텐츠의 처리속도를 향상 시킬 수 있는 바인딩 기법을 제안하였다. 제안된 바인딩 기법은 상위 계층에 JSR184의 표준 인터페이스를 마련하여 기존 컨텐츠와 호환을 이룰 수 있도록 하였고, 하위 계층에 OpenGL/ES의 표준을 구현하여 처리속도가 향상되도록 한 후, 중간 계층에서 이기종 코드 변환 언어인 KNI를 사용한 바인딩 구현을 통하여 자바 기반 삼차원 엔진의 성능을 향상시켰다.

이와 같이 바인딩 구현은 자바를 기반으로 한 품질 좋은 삼차원 컨텐츠의 엔진을 좀 더 빠른 속도로 구동 시킬 수 있고, 순수 JSR184보다 적은 메모리가 사용되어, 휴대용 임베디드 기기의 설계에서 요구되는 특성을 갖추고 있다. 또한 하드웨어와 연결할 수 있는 하위 부분의 바인딩 구현은 해당 프로세서에 최적화 할 수 있도록 어셈블리 언어로 부분 변경이 가능하여 현재 구현된 C 형태보다 성능을 향상 시킬 수 있다[13].

참 고 문 헌

[1] JSR-184 Expert Group, "JSR-000184 Mobile 3D Graphics API for J2ME," http://www.jcp.org/en/jsr/detail?id=184.

[2] The Khronos Group, "OpenGL ES Overview," http://www.khronos.org/opengles/.

[3] Sun Micro Systems, "KVM Porting Guide 1.0," SUN, Palo Alto, 2000.

[4] JSR-239 Expert Group, "Java bindings to the OpenGL ES(Embedded Subset) native 3D graphics library," http://www.jcp.org/en/jsr/detail?id=239.

[5] Tomas Moller and Eric Haines, *Real-Time Rendering*, A K Peters, Ltd., Massachusetts, 1999.

[6] The Khronos Group, "OpenGL 1.3, Reference Manual," http://opengl.org/documentation/.

[7] The Mesa Project, "The Mesa 3D Graphics Library," http://www.mesa3d.org.

[8] D. Kirk, "Unsloved Problems and Opportunities for High-Quality, High-Performance 3-D Graphics on a PC Platform," *Proceedings of SIGGRAPH-Eurographics Workshop on Graphics Hardware*, Keynote talk, 1998.

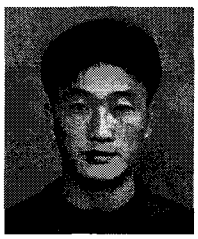
[9] Sun Micro Systems, "SUN Wireless Toolkit 2.2 Release," http://java.sun.com/products/sjwtoolkit.

[10] Kishonti Informatics LP, "JBenchmark Community," http://www.jbenchmark.com.

[11] Compag, "Neon: A (Big) (Fast) Single-Chip 3D Workstation Graphics Accelerator," 1999.

[12] Hyun-Jea Woo, Jong-Chul Jeong, and Moon-Key Lee: "The Design of efficient 3D graphics Rendering Architecture for mobile device," *Proc of KGS*, pp. 337-324, 2000.

[13] M Kameyama, "3D Graphics LSI Core for Mobile Phone Z3D," *Graphics Hardware*, pp. 60-67, 2003.



김 영 옥

2000년 2월 한국방송통신대학교 (전자계산학 이학사)

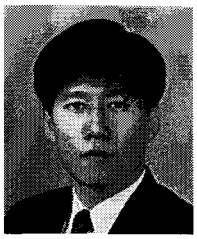
1999년 10월~2001년 1월 위넷컴 대표이사

2001년 1월~2002년 5월 (주)지엔비커뮤니케이션 연구 1팀 개발실장

2002년 12월~2004년 2월 IMT SOFT 선임연구원

2004년 3월~현재 (주)코아로직 책임연구원

관심분야 : 임베디드시스템, 이차원/삼차원 그래픽스, 모바일 자바



노 영 섭

1988년 2월 인하대학교 전자공학과(공학사)
1996년 8월 한국과학기술원 정보및통신공학과(공학석사)
2005년 2월 고려대학교 전기, 전자, 전파공학과(공학박사)
1987년 11월 1998년 2월 LG전자 미디어통신연구소 선임연구원
1998년 3월~2001년 2월 청강문화산업대학교 이동통신과 교수
2001년 3월 2005년 2월 주식회사 싸이버뱅크 연구개발부문 상무이사
2005년 3월~현재 서울벤처정보대학원대학교 임베디드시스템학과 교수
관심분야 : 임베디드시스템, 모바일 컴퓨팅, 이동통신, 유비쿼터스 네트워크