모바일 3D 게임 엔진을 위한 효율적인 스킨드 메시 처리

조종근°
숭실대학교 컴퓨터학과°*
jkdang@empal.co.kr

An Efficient Skinned-Mesh Process For Mobile 3D Game Engine

Jong-Keun Cho°
Dept. of Computer Science, Soongsil University,

요 약

기존에는 모바일 표준 3D 그래픽 API(C언어 기반)인 OpenGL-ES를 사용하여 모바일 3D 게임 엔진을 제작해, 핸드폰에 애플리케이션을 작동시켰으나, 저수준(Low-Level)의 다양한 기능만 제공함으로써, 다양한 콘텐츠 제작 및 콘텐츠 호환성에 제약이 많았다. 이에 본 논문에서는 OpenGL-ES보다 더욱더 다양한 고수준(High-Level)의 API를 제공하면서도 GSM 폰을 중심으로 J2ME상에서 자바환경에 최적화된 모바일 표준 3D API(Java언어 기반)인 JSR-184로 모바일 3D 게임 엔진을 제작한다. 또한, 스킨드 메시(Skinned-Mesh) 형태를 가지는 3D 모델의 처리속도를 향상시키는 방법을 제시하고, 실험 결과로 엔진의 성능을 증명해 보인다.

ABSTRACT

The game engine has executed an application after making a mobile 3D game engine which is based on mobile 3D standard graphic API using openGL-ES so far. But, We could not do it satisfactorily that contents compatibility of various types as a various low-level's function is supported.

At this point, This study introduce a mobile 3D game engine which is based on mobile 3D standard graphic API using JSR-184 that supporting a high-level's API more than openGL-ES and optimizing to Java environment on J2ME in the center of GSM phone. Also, We shows that the proposed skinned-mesh scheme for enhancing the process speed of a 3D object on JSR-184 engine. The experimental results are shown.

Keyword : JSR-184, Skinned Mesh, Mobile 3D Game Engine

접수일자 : 2008년 10월 07일
일차수정 : 2008년 11월 03일
심사완료 : 2008년 11월 06일

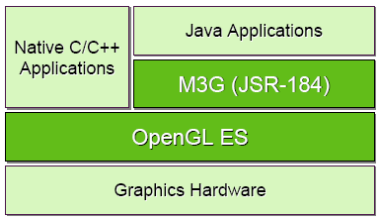
1. 서 론

최근 모바일 3D 업계에서 OpenGL-ES와 더불어 주목받고 있는 것이 JSR-184이며, 자바환경에서 최적화된 모바일 표준 3D 그래픽스 API이다. J2ME 환경에서 3차원 그래픽을 구현하기 위해서 저수준(Low-Level)의 OpenGL을 이용할 경우, 코드가 길어져 MIDlet (MIDP Application)의 덩치가 커지므로 속도가 느려질 수밖에 없고, 자바 3D API를 이용할 경우에는 스펙의 양이 너무 방대하기 때문에 역시 MIDP(Mobile Information Device Profile)에 이용하기엔 적합하지 않다.

본 논문에서는 다양한 모바일 3D 멀티콘텐츠 수용을 위해 JSR-184를 이용해서 J2ME 기반의 모바일 3D 엔진을 설계 및 구현하였다. 또한 노키아(Nokia)의 JSR-184 엔진에서 스킨드 메시(Skinned-Mesh) 형태의 모델을 처리하는 것보다 좀더 빠르게 데이터를 처리할 수 있는 방법을 제시한다.

2. 관련 연구

최근 노키아의 JSR-184 모바일 3D 엔진을 포함한 플랫폼 구성도는 [그림 1]과 같은 구조로 작성되었다.[8] OpenGL-ES 단에 자바 가상머신과 M3G (JSR-184) 단에 자바 인터페이스(GUI)가 구동되어 실행되는 구조로 본 논문에서 설계한 방법과 크게 다르지 않다.

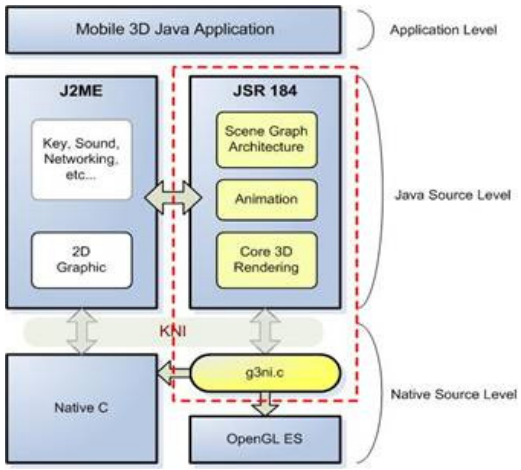


[그림 1] 노키아 JSR-184 3D 플랫폼 구성도

3. JSR-184 모바일 3D 엔진구성도

본 논문에서 구현한 JSR-184 모바일 3D 엔진 구성도는 다음과 같다.[1][2]

[그림 2]에서 보는 것처럼 JSR-184의 30개 클래스(Class)들과 250개 메소드(Method)들로 3D 처리 부분을 구현해 놓고, KNI(K-Native Interface)를 이용해서 C언어로 OpenGL-ES 함수와 매핑시켜 동작하게 한다.[3]



[그림 2] 제안하는 JSR-184 3D 엔진구성도

3.1 자바언어 구현부분

JSR-184 스펙에 정의된 클래스와 메소드들은 순수 자바언어로 구현한다. 여기에 해당되는 것들은 다음과 같다.

3.1.1 3차원 그래픽스 부분

모델링(Modeling), 변환(Transformation), 카메라(Camera), 텍스쳐 매핑(Texture Mapping), 재질(Material), 라이팅(Lighting) 등의 기본적인 3D 그래픽스 표현과 관련된 부분이며, 기존의 OpenGL-ES 엔진을 최대한 활용할 수 있도록 자바 래퍼(Wrapper)를 작성하여, 3차원 그래픽스 부분을 구현하였다.[4]

3.1.2 애니메이션 부분

키 프레임(Key Frame) 애니메이션 및 본(Bone) 애니메이션, 모션모핑 등을 의미하며 이 부분은 OpenGL-ES에 없는 부분으로써 먼저 C언어로 구현한 후, 이를 자바환경에서 사용할 수 있는 자바 래퍼를 별도로 작성하였다.

3.1.3 씬그래프(Scene Graph) 부분

계층구조(Hierarchy)지원, 오브젝트 트래버스(Traverse), 피킹(Picking), 정렬(Alignment) 기능은 자바로 구현하였다.

3.2 자바언어와 C언어 연결부분

J2ME 구현방식과 마찬가지로 JSR-184 역시 실제동작은 대부분 C언어로 작성된 부분을 통해서 동작한다. 즉, 자바에서 구현된 데이터 값들을 C언어로 작성된 부분으로 넘겨줄 때 이들 구조에 대한 적절한 변환을 수행하는 부분으로 KNI (K-Native Interface)를 사용하여 자바와 C언어 부분을 연결부분을 구현하였다.

3.3 C언어 구현부분

자바로부터 KNI를 통해 넘겨받은 그래픽 관련 데이터들을 적절한 OpenGL-ES 함수와 매칭시켜 JSR-184 랜더러의 실제 동작부분을 구현하였다.

4. 설계 및 구현

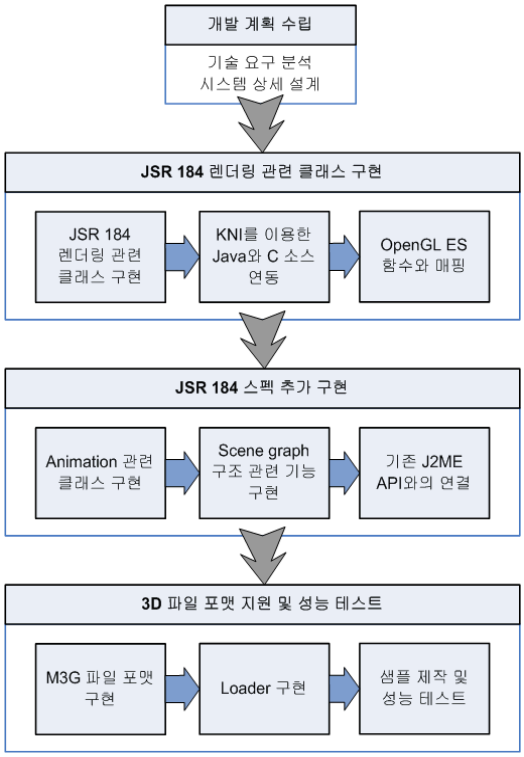
본 논문 3장의 내용들을 기반으로 설계한 JSR-184로 엔진구현을 제작하기 위해 [그림 3]과 같은 절차를 따라서 작성하였다.

4.1 전체적인 설계도

본 논문에서 사용된 파일 포맷은 JSR-184에서 제공하는 M3G 파일 포맷을 사용하여 샘플 파일을 작성한 후 실험에 사용하였다.

[그림 3]에서 중요한 부분인 서브메시 랜더링(SubMesh Rendering)을 위한 render() 메소드 경우 다음의 7단계 절차대로 수행한다.

1) rendering 여부 검사
2) Light 설정 적용



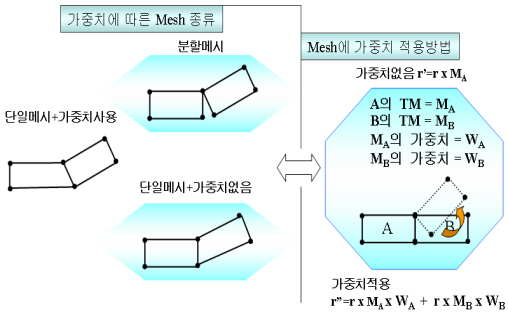
[그림 3] JSR-184를 이용한 3D 엔진구현과정

3) Camera Transform 설정된 View Trans-form 적용
4) 각종 Appearance 관련 정보
   CompositingMode ,PolygonMode, Texture2D, Fog, Material 등 적용
5) Vertex 관련 정보
   Vertex Position, Color, Normal, Texture Coordinate 등 등록
6) Model Transform(World Coordinate내에서의 Object Transform) 정보 설정

7) Rendering 실행

4.2 스킨드 메시 모델 처리 향상을 위한 방법

스킨드 애니메이션에서는 하나의 메시(Mesh)를 사용하고, 여러 번의 변환 가중치 변화에 따른 가중치 결과를 혼합하여 사용하며, 정점(Vertex)들은 영향을 받는 뼈와 가중치를 가지고 있어야 한다. 단, 가중치의 합은 1(100%)이다.



[그림 4] 스킨드 애니메이션 결과값 비교

[그림 4]를 보면 알 수 있듯이, A와 B라는 부분의 애니메이션이 달라질 경우, 분할 메시의 경우에는 별다른 방법이 없는 것을 알 수 있다. 그러나, “단일 메시+가중치 없음”의 경우에는 비교적 그럴듯한 결과 값을 얻을 수 있으나 관절부위가 자연스럽지 못한 것을 알 수 있다. 마지막, “단일 메시+가중치 사용”의 경우에는 가장 이상적인 형태로 A의 애니메이션 행렬 MA와 B의 애니메이션 행렬 MB가 각각 가중치 WA와 WB값에 의해서 결합된 후, 최종값을 누적해서 적용되는 것을 알 수 있다.

이와 같이 연결 부분의 정점은 두개의 좌표계를 함께 사용하며, 각각의 결과를 적절히 혼합하여 사용하며, 이를 수식으로 표현하면 수식 1과 같다.[5]

$$\begin{pmatrix} Rx \\ Ry \\ Rz \end{pmatrix} = WM \begin{pmatrix} Ax \\ Ay \\ Az \end{pmatrix} + (1-W)M \begin{pmatrix} Bx \\ By \\ Bz \end{pmatrix}$$

수식 1. 스킨드 메시 적용 행렬계산

수식 1에서 W는 정점 가중치를 의미하며, 해당 정점이 양쪽 좌표계에 동일하게 의존할 경우 각각 W=0.5를 적용한다, 한편, 뼈대 다른 끝 부분의 경우 정점 가중치는 0 또는 1이 되어 하나의 좌표계에만 의존하게 된다. 결국 전체 정점의 개수만큼 루프를 돌면서 각각의 정점마다 4개씩의 블렌드 가중치(Blend Weight) 연산을 한다. 이때, 영향을 받는 뼈대와 인덱스에 기반해서 매트릭스 팔레트로부터 스킨 행렬을 얻어오고, 이 행렬과 정점을 곱한 값을 가중치만큼 더해주는 것이다. 이를 수식으로 표현하면 수식 2와 같다.

$$V_{world} = V_{local} * M[INdex0] * W_0 + V_{local} * M[INdex1] * W_1 + V_{local} * M[INdex2] * W_2 + V_{local} * M[INdex3] * W_3$$

수식 2. 스킨드 메시 적용수식

JSR-184는 스킨드 메시에서 addTransform (Node bone, int weight, int firstVertex, int numVertice) 함수를 통해 몇 번째 정점(Vertex)부터 몇 번째 정점이 어느 본(Bone)의 가중치 영향을 얼마만큼 받는지 결정되는데, addTransform()은 여러 번 일어날 수 있으며, 그 횟수는 제안되어 있지 않다. 이때, Wi는 각 Vertex에 영향을 미치는 모든 노드(Node)의 가중치에 대한 정규화 값이어야 하며, 같은 본에 대해 addTransform()이 여러 번 발생할 수도 있는데, 이때 해당 범위의 정점에는 가중치 값이 누적되어야 한다.

4.2.1 노키아 스킨드 메시 적용 방법

노키아 경우 WeightedTransform이라는 inner class 이용하여 Skinned-Mesh를 구현하고 있다.

addTransform() 함수가 발생할 때마다 하나의 Weighted-Transform 오브젝트를 생성하여 해당 본의 “rest” 상태의 Transform, Weight에 영향을 받는 정점들의 첫 번째 인덱스와 마지막 인덱스 등의 정보를 정하게 되고, 생성된 Weighted-Transform 오브젝트는 리스트에 추가시킨다. 이렇게 저장된 정보는 렌더링시에 각 정점별로 모든 WeightedTransform 리스트를 검사하여 현재 정

점이 포함된 경우(WeightedTransform에 저장된 첫 번째와 마지막 정점 인덱스 사이에 현재 정점 인덱스가 포함)의 WeightedTransform들을 찾고, 해당 WeightedTransform들의 가중치 합을 구한 후, 다시 각각의 WeightedTransform의 가중치들을 그 합으로 나눔으로써 가중치 정규화를 계산하는 비효율적인 구조로 이루어져 있다.

또한 본의 개수만큼만 생성하면 될 정점의 개수보다 많은 Position Transform, Normal Trans- form, 그리고, 렌더링 타임에서 inner class의 개수만큼 생성하고 Position Vertex 값과 Normal Vertex 값을 계산하여 누적한다. 즉, Position Trans- form과 Normal Transform은 현재 상태의 Vertex Transform이 반영되어야 하므로 렌더링 타임에 매번 계산하여 구해야 하는 정보이다.

요컨대 이렇듯 노키아의 구현 방식은 가중치의 정규화 방법이 비효율적일 뿐만 아니라 addTran- sform() 함수가 많이 일어날수록 메모리 사용량이 늘어날 뿐만 아니라 렌더링 시간이 많이 소요된다.

4.2.2 제안하는 스킨드 메시 적용 방법

본 논문에서 제안하는 알고리즘의 경우 이러한 문제점을 보완하고 효율적인 Skinned-Mesh의 렌더링을 위해 다음과 같이 구현하였다. 각 정점이 어느 본에 대해 얼마만큼의 가중치 영향을 받는지에 대한 정보를 저장할 가중치 리스트를 2차원 배열(Weight List[Vertex Number][Bone- Number])로 만들고 각 정점별로 모든 본의 가중치 합을 저장하기 위한 배열 Weight Sum- [Vertex Number]를 두었다. 이들의 정보는 모두 0으로 초기화된 상태에서 add- Transform() 함수가 일어나는 순간에 어느 범위의 정점이 어느 본에 대해 얼마만큼의 영향을 받는지에 대한 가중치 정보를 해당 위치에 누적하고, 갱신이 이루어진 정점에 대해서는 weightSum을 업데이트함으로써 가중치의 합도 렌더링 타임이 아니 add- Transform() 함수가 일어나는 로딩 타임에 미리 계산을 해두도록 하였다.

이런 구조를 가질 때, 렌더링 타임에서는 본의 갯수만큼에 해당하는 Position Transform과 Normal Transform만 계산해주면 된다. 또한 각 정점별 가중치의 합을 미리 계산해 둔 상태이기 때문에 이 과정은 렌더링 타임에서 생략할 수 있으며, 특정 본에 대한 가중치가 0일 경우에는 해당 본의 영향을 받지 않는다는 뜻이므로 계산을 생략함으로써 불필요한 연산을 막을 수 있다.

4.3 스킨드 메시 적용 알고리즘

본 논문의 스킨드 메시 오브젝트에 적용한 알고리즘은 다음과 같다.

기본적으로 스킨드 메시 오브젝트 처리시 [그림 4]와 같은 계산이 실시간으로 처리되어야 정확한 움직임에 자연스러운 메시와 메시 사이의 연결부분을 적용할 수 있다.

```
 //Deform 관련 정보저장
int                 weightList[vertex 개수][bone 개수]
int                 weightSum[vertex 개수]
Transform       toBone[bone 개수]
Transform       positionTransform[bone 개수]
Transform       normalTransform[bone 개수]
Vector             boneList

addTransform() {
  해당 위치의 weightList에 weight값 누적
  weightSum 누적하여 새로 계산
}
morph()
{
    for(int i = 0; i < bone 개수; i++)
    {
      positionTransform 계산
normalTransform 계산
}
     for(int i = 0; i < vertex 개수; i++)
     {
          for(int j = 0; j < bone 개수; j++) {
             weightSum과 transform을 이용하여
             vertex position과 normal deform
      } } }
```

즉, 각각의 정점(Vertex)들은 어느 본에 대해 얼마만큼의 가중치 영향을 받는지에 대한 정보와 각각의 정점(Vertex)별로 모든 본의 가중치 값들을 실시간으로 계산하여 렌더링(Rendering)시마다 계산된 값을 적용하는 것이 아니라, 최종 로딩시에 미리 계산된 값을 한번만 적용해서 처리하도록 하여, 노

키아 JSR-184 엔진의 스킨드 메시 적용 알고리즘에서 가중치 값을 정규화할 때 비효율적이고 메모리 사용량이 늘어나며 렌더링 시간이 늘어나는 문제점을 개선하였다.

5. 실험 방법

본 J2ME 환경에서 제공하는 에뮬레이터 상에서 3D MAX 7.0으로 작업한 샘플을 M3G 익스포터(Exporter)를 이용해 파일로 추출한 후, JSR-184로 구현한 엔진으로 성능 테스트를 한다. 요컨대 JSR-184 표준 파일포맷인 M3G 파일형태로 익스포트해서 사용한다. 실제로 엔진성능을 위해 사용된 3D 오브젝트는 Skinned-Mesh로 표현하였고 Skinned-Mesh 애니메이션을 적용하였으며, 캐릭터에 텍스처를 입히고 카메라를 적용하였다.

[표 1]은 노키아와 제안하는 JSR-184 엔진에서 사용된 3D 샘플 오브젝트이다.

[표 1] 실험에 사용된 3D 샘플1(스킨드 메시 오브젝트)

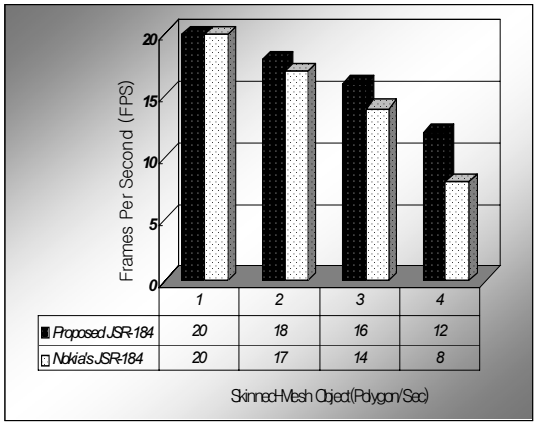
JSR-184 TYPE	물체	파일 포맷
폴리곤수1	100 Polygon/Sec	Arm.M3G
폴리곤수2	350 Polygon/Sec	Boy.M3G
폴리곤수3	500 Polygon/Sec	DragonFly.M3G
폴리곤수4	1200 Polygon/Sec	Monster.M3G

6. 실험 결과

[그림 5]는 JSR-184로 구현한 엔진으로써 3D MAX 7.0을 사용해서 M3G 파일 포맷으로 스킨드 메시를 적용해서 제작한 소년이 스케이트 보드를 타고 묘기부리는 모습을 실행한 화면이다.

[그림 5] JSR-184 엔진을 적용한 실행화면

[그림 6]의 결과는 노키아와 제안하는 JSR-184 엔진으로 표 2의 형태로 작성한 3D 샘플 스킨드 메시 오브젝트를 테스트한 성능수치이고, 노키아 JSR-184의 엔진보다 1~4FPS 정도의 성능이 향상된 것을 볼 수 있으며, 스킨드 메시로 작성된 폴리곤수가 많은 오브젝트일수록 노키아의 JSR-184 엔진보다 빠르게 처리한다는 것을 알 수 있다.



[그림 6] 노키아 VS 제안하는 JSR-184 엔진 성능

요컨대 Handheld 장치들의 연산 처리 속도가 PC보다 상대적으로 낮은 환경에서 게임 애플리케이션이 동작한다는 점을 감안해 보면 성능향상이 더욱더 필요하다. 이런 측면에서 본 논문은 의의가 있다고 생각한다.

7. 결 론

본 논문에서는 J2ME 환경에서 모바일 3D 그래픽스 표준인 JSR-184를 이용하여 유럽을 중심으로 미국과 중국 등의 전 세계 이동통신 시장의 70%를 차지하고 있는 GSM 폰 시장에서 상용화할 수 있는 모바일 3D 엔진을 설계 및 구현하였다. 또한 노키아(Nokia)의 JSR-184 엔진에서 스킨드 메시(Skinned-Mesh) 형태의 모델을 처리하는 것보다 좀 더 빠르게 데이터를 처리할 수 있는 방법을 제시하였다.

향후 연구방향으로는 JSR-184에서 처리되는 모든 그래픽 처리단계에서의 최적화에 관한 기법들에 대해 연구할 계획이다.

8. 참고 문헌

[1] 조종근, “G3SDK를 이용한 모바일 3D 프로그래밍,” 고미드, 2005.
[2] 고미드, http://www.gomid.com
[3] 조종근, 박윤희, 김종민, “J2ME상에서 JSR-184를 이용한 모바일 3D 엔진의 설계 및 구현,”한국정보과학회학술지 제32권, 제2호, pp.673-675, 2005.10.
[4] http://www.forum.nokia.com/java/jsr184
[5] 김용준, “3D 게임 프로그래밍,” 한빛미디어, 2004.
[6] 조종근, 김종민, “OpenGL-ES 기반의 모바일 3D 블루투스 엔진 설계 및 구현,” 한국게임학회 논문지 제6권, 제1호,pp.23-28, 2006.3.
[7] James.Dong.L and Twigg.Christopher.D, "Skinning Mesh An Animations," ACM, Vol.24,No.03, pp.399-407, 2005.7.
[8] Enrico.Gobbetti and Fabio.Marton, "Far voxels:a multiresolution Framework for Interactive Rendering of Huge Complex 3D Model on Commodity Graphics Plat-forms," ACM Trans on Graphics, Vol.24, No.03, pp.878-885, 2005.7.
[9] J.W.Muchow, "Core J2ME Technology &MIDP," Prentice Hall, 2001.
[10] J2ME, http://java.sun.com/j2me
[11] 크로노스그룹, http://www.khronos.org

조 종 근

1998년 성결대학교 컴퓨터공학(공학사)
2001년 숭실대학교 컴퓨터학과(공학석사)
2004년 숭실대학교 컴퓨터학과(공학박사)
2004년～현재 모바일 3D 표준화포럼/전문위원
2004년～현재 (주)GOMID/수석연구원

관심분야 : 모바일 컴퓨팅, 멀티미디어, 컴퓨터 그래픽스