자바를 이용한 2.5D 메타버스 게임 엔진 구현

한승훈°, 김은주°
°한림대학교 컴퓨터공학과

Implementation of 2.5D Metaverse Game Engine Using Java

Seung-Hoon Han°, Eun-Ju Kim°
°Dept. of Computer Engineering, Hallym University

요 약

자바의 특징 중 코드 재사용성을 강조하여 게임 엔진에서의 코드 재사용과 확장성을 고려한 비행기 전투 게임인 메타버스 게임 엔진을 개발한다. 현재 PC 게임 개발은 C/C++과 DirectX 를 사용한 게임 개발을 주로 하고 있다. 제한된 개발 언어의 사용은 게임 엔진 제작에 있어 게임 엔진의 다양화를 억제하는 요소가 된다.

1. 서론

1995 년 자바가 인터넷에 처음 등장하면서 자바를 이용한 포커, 고스톱 등 카드형식의 웹 게임이나 간단한 애플릿을 이용한 PC 게임이 개발되었다. 당시의 하드웨어로는 느린 처리 속도와 용량이 큰 JVM(Java Virtual Machine:자바 가상 머신)을 따로 설치해야 하는 불편함이 있었기 때문에 자바로 PC 게임을 개발하는 개발자 수요가 점점 감소하게 되었다. 이후에 자바 게임 시장은 자바 언어가 시스템 플랫폼에 독립적이라는 장점을 이용하여 J2ME(Java 2 Platform Micro Edition) 와 WIPI(Wireless Internet Platform for Interoperability : 무선 인터넷 표준 플랫폼)등 모바일 게임 분야로 개발 시장이 변화하게 되었다. 현재는 하드웨어와 JVM 의 성능이 좋아져 자바로 PC 게임을 개발 할 수 있는 환경이 조성되었다. 하지만 현실적으로는 C/C++과 DirectX, OpenGL(Open Graphics Library) 등의 기본 개발 환경을 그대로 사용하고 있는 실정이다. 그 동안 축적된 게임 개발 방법 및 Know-How, 경험 등에 의해 자바로 개발하려는 시도가 많지 않기 때문이다.

본 연구는 2006 년 지방대학 혁신역량 강화사업(NURI) 문화 콘텐츠(CT) 인력양성 2 차년도 사업의 연구비 지원으로 수행되었습니다.

본 논문은 자바를 이용한 게임 엔진으로 비행기 전투게임을 구현하였다.

2. 자바 게임 엔진

게임 엔진이란 게임을 수행하기 위한 핵심 기술을 말한다. 게임엔진은 게임내의 물리적 움직임이나 그래픽 엔진, 사운드 엔진, 네트워크, NPC(Non-Playable Character:플레이어를 제외한 캐릭터) 처리, AI(Artificial Intelligence:인공지능), 오브젝트 처리 등을 총괄적으로 관리하여 게임을 쉽게 제작 할 수 있게 한다. 게임 엔진을 만들기 위해서는 하드웨어 가속을 지원하는 빠른 처리 속도와 네트워크, input, 물리 엔진, 알고리즘, 사운드 등의 다양한 라이브러리의 지원과 엔진의 편의성 및 확장성을 고려한 조건이 내포되어야 한다.

메타버스(Metaverse) 엔진은 자바의 특성을 최대한 활용하였으며 게임 엔진의 확장과 편의성에 중점을 두어 제작을 하였다. 코드의 재사용과 기능 확장을 쉽고 간단하게 하기 위해 디자인 패턴에 기반을 두고 게임 엔진을 설계하였다.

J2SE(Java 2 Standard Edition) 5.0 플랫폼과, Eclipse 3.1 의 개발 툴, Windows XP Professional 운영 체제, Swing/AWT /Java2D/OpenGL(JOGL) 등의 라이브러리를 사용한 환경에서 게임을 구현하였다.

3. 메타버스 게임 엔진

3.1 메타버스 엔진의 시스템 구성

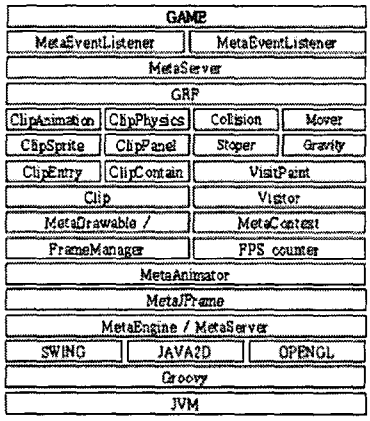


그림 1. 시스템 구조

그림 1 은 메타버스 시스템의 구조를 보여 주고 있다. 이 중 MetaJFrame 은 Swing 의 JFrame Class 를 상속받은 클래스로 주로 게임의 FullScreen 제어나 UI(User Interface) 컴포넌트(버튼등의 구성 요소)의 Container(컴포넌트들을 담는장소) 역할을 한다.

MetaAnimator 는 Frame rate 와 게임 Thread(동시에 처리할수 있는 프로그램의 단위)를 관리하는 클래스와 게임 프레임 레이트와 각종 스레드를 관리하는 클래스이다. 고정프레임(일정 시간을 고정시켜 수행하는 방식) 가변 프레임(처리량에 따라 나타내는 수행하는 방식) 등을 모두 사용할 수 있으며, 네트워크 동기화(네트워크에서 서로 다른 컴퓨터의 상태를 똑같이 유지하는 방법)에도 사용 될 수 있다.

MetaDrawable 은 각종 Render 를 정의할 수 있으며 OpenGL, JAVA2D, Swing 등을 메타버스 엔진에 맞게 수정하여 사용할 수 있는 인터페이스이다. 자바 자체 하드웨어 가속 기능을 사용할 수도 있고 JOGL 을 사용하여 메타버스 엔진과 OpenGL 을 연동하여 사용할 수 있다.

Clip 은 게임상의 객체를 정의 하기 위한 최상위 추상 클래스이다. 이 클래스는 디자인 패턴의 Composite 패턴을 사용하여 설계를 하였다. 만약 게임 내의 객체를 만들고자 할 때 Clip 클래스를 상속하면 게임 엔진 내에서 바로 사용 할 수 있는 클래스로 되는데 이것은 새로운 클래스를 정의하는 매우 간단한 방법이다. 또한 각종 Linked List, Array, Map, Hash Table 등의 자료구조를 Clip 을 상속받은 클래스를 통해 객체들을 필요에 따라 원하는 자료구조로 관리를 할 수 있다.

Visitor 는 게임상의 객체들의 제어나 움직임을 위한 인터페이스이다. 재귀함수의 특징이 있는데, 간결하고 분석하기 편한 코드를 만드는데 도움을 주기도 한다. Visitor 는 물리 엔진의 충돌 체크와 속도, 가속도 처리, 사용자 정의 렌더링를 위한 객체의 그리기 순서 정의 및 길 찾기 알고리즘, 게임 메시징 처리를 한 개의 인터페이스로 정의하여 코드의 재사용성을 높일 수 있는 인터페이스이다.

GRF 는 그림 파일을 저장하기 위해 효율적인 자료 관리를 해주는 자체 저장 방식이다. 이는 이미지 파일 처리에 있어서 다량의 고속 이미지 처리를 위해 자바에 내장된 API 를 사용하지 않고 GRF 코덱을 자바로 포팅해서 사용하였다. GRF 는 비손실 압축 알고리즘인 LZ77 알고리즘을 사용한 Indexed Image(팔렛트 정보를 가지는 이미지)이다. 또한 이미지를 한번에 모두 불러 쓰는 것이 아니라 실시간으로 이미지 raster 정보를 읽어오는 방식을 사용한다. 대량의 이미지를 불러올 때 속도가 느려지는 현상을 없앨 수 있다.

MetaEventListener 는 실제 구현되는 게임의 코드를 정의하는 부분이다. 고정 프레임과 동적 프레임을을 마음대로 적용할 수 있기 때문에 게임에 맞는 코드를 직접 제작 할 수 있다. 이 부분은 게임에 필요한 코드들만 담고 있기에 MetaEventListener 를 객체 직렬화하면 어떠한 게임이라도 게임의 save/load 기능을 간단하게 구현 할 수 있다.

MetaServer 는 네트워크 게임지원을 위해 네트워크 서버를 같이 제작 하였는데 대화방 방식의 게임 서버용으로 XML-RPC 를 사용하여 게임 프로토콜의 확장과 수정이 용이하게끔 하였다

3.2 메타버스 엔진의 구현

메타버스 엔진의 구현은 코드 재사용을 목적으로 세 부분으로 나누어 개발 하였다.

3.2.1 게임내의 객체 정의 부분

객체 정의란 게임상의 건물, 비행기 등과 같은 객체를 정의하는 부분으로 Clip 추상클래스를 통해 그림 3 과 같이 게임에서 필요한 객체를 정의한다. Clip 추상클래스는 기본적으로 필요한 객체의 좌표들을 저장하고 비행기 클래스를 생성하기 위해서 물리적 특성을 주는 물리적 특성이 적용된 ClipPhysics 클래스를 상속받아 사용한다.

게임 내에서 객체의 행동을 처리하기 위해서는 객체의 특성마다 각각 행동 단위를 따로 구현하여야 하는데 코드의 복잡성을 증가시킬 뿐만 아니라 내부 구조가 비효율적인 게임이 된다. 이점을 보안 하기

위해 각 처리의 행동요소를 분리하여 사용하는 개발 방법을 적용하였다. 그림 4 는 게임 내에서 비행기의 행동을 처리 하기 위한 간단한 예로 비행기의 움직임을 처리하는 부분과 그 비행기의 중력을 처리하는 부분을 따로 구현한 모습이다. 이러한 클래스의 구분은 코드 재사용성을 높이기에 충분하다.

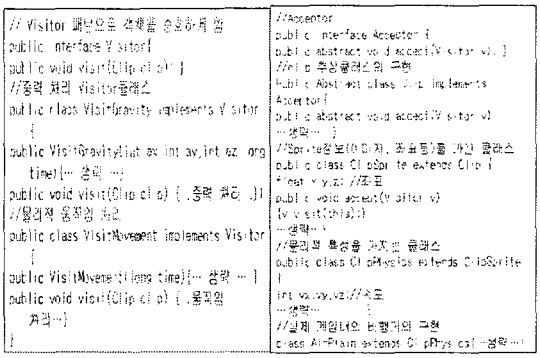


그림 2 객체의 행동 처리 부분 및 객체의 행동 및 처리 프로그램

3.2.2'메인 구현 부분

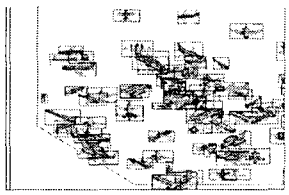
```
AirPlane airPlane = new AirPlane();
airPlane.accept(new VisitGravity(0,1,0,10));
airPlane.accept(new VisitMovement(10));
```

그림 3. 메인 프로그램

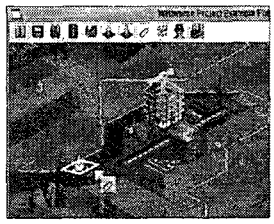
실제 메인 게임에서는 그림 5 와 같이 간단하게 물리엔진을 구성을 할 수 있으며 행동처리에 대한 부분이 클래스로 따로 분리 되었기 때문에 코드 재활용이 가능하였다. 물리 엔진뿐만 아니라 각종 길찾기, 렌더링, 네트워크 동기화 등과 같은 알고리즘 구성에도 효과적으로 재사용 할 수 있다.

3.3.3 구현 화면

그림 6 은 실제 구현한 게임 화면이다. (a)는 물리엔진을 구현하여 각종 객체들을 움직이며 테스트 하는 화면이며 (b)는 비행기 게임의 배경이 되는 맵을 제작하는 에디터이다. 맵의 자료구조를 그대로 비행기 게임에 사용하고 (c)의 비행기 게임은 맵 에디터의 자료구조를 불러와 비행체를 띄우는 작업만 하면 되므로 이는 게임 개발에 있어 코드 재사용성을 극대화 하였다고 할 수 있다.



(a) 물리엔진 테스트 화면



(b) 비행기 게임 멥 에디터

(c) 비행기 전투 게임

그림 4 구현된 메타버스 게임 화면

4. 결론 및 향후 연구 과제

현재 C/C++을 사용한 게임 엔진의 대안으로 자바 게임 엔진을 제시하였다. 이 게임 엔진은 코드 재사용성 등의 이점을 통해 게임 개발 자료의 공유를 보다 활성화 시킬 수 있으며 이는 게임 개발의 생산성 향상을 보여주고자 하였다. 하지만, 게임 개발이 단기간, 적은 인력으로 완성되는 것이 힘들기 때문에 많은 적용사례를 보이지 못했다.

그리고 엔진을 개발 하며 여러 문제점이 발견이 되었는데 그 중에 가장 먼저 해결 해야 할 문제는 가비지 컬렉션에 의한 속도 저하로 게임 내 메모리 관리 방식을 개선할 필요가 있었다. 또한 게임 개발에 있어 다양한 게임 관련 라이브러리가 자바에서는 절대적으로 필요하고, 게임을 보다 쉽게 개발하기 위해서는 게임 제작 에디터나 스프라이트 툴 등과 같은 개발 툴들의 개발이 필요하다.

5. 참고문헌

[1] Java GENGO DE MANABU Design Pattern NYUMON Softbank Publishing.INC, Hiroshi Yuki

[2] GRF Codec, http://www.ttdpatch.net/grfcodec/

[3] LZ77 Algorithm, http://en.wikipedia.org/wiki/LZ77

[4] Physics for Game Developers, O'Reilly, David M. Bourg

[5] Beginning Math and Physics for Game Programmers, New Riders Publishing, Wendy Stahler

[6] Java 1.4 Game Programming, Wordware Publishing, Andrew Mulholland,Glenn Murphy