http://dx.doi.org/10.7583/JKGS.2012.12.4.13

실시간 다중 렌더링을 위한 이중 서버 구조※

임충규
성공회대학교 멀티미디어시스템공학과
cglim@skhu.ac.kr

A 2-Tier Server Architecture for Real-time Multiple Rendering

Choong-Gyoo Lim
Dept. of Multimedia System Engineering, Sungkonghoe University

요 약

초고속 인터넷의 광범위한 보급으로 가능해진 클라우드 컴퓨팅 기반 게임 서비스는 클라우드 노드에서 게임을 실행하고 게임의 영상을 원격 사용자의 단말기에 영상 스트림으로 전송함으로써 게임 서비스가 이루어진다. 사용자 입력은 게임에 즉각적으로 전송되고 반영된다. 이러한 서비스가 가능한 이유는 사용자 입력이 반영되고 게임 영상이 사용자에게 전달되는데 걸리는 시간이 최소화되어, 컴퓨터 게임에서 요구되는 반응성을 일반적으로 만족시킬 수 있었기 때문이다. 하지만 이러한 서비스는 고품질 3D 게임을 서비스하는 경우, 서버 구축에 많은 비용이 소요될 수 있다. 클라우드 노드가 탑재하고 있는 일반적인 그래픽 시스템은 동시에 하나의 3D 어플리케이션을 지원하도록 설계되어 있기 때문이다. 하나의 클라우드 노드에서 다수의 3D 게임을 실행하기 위해서는 그 실행에 필요한 실시간 다중 렌더링 기술이 필수적이다. 본 논문은 다수의 컴퓨터 게임을 하나의 클라우드 노드에서 실행시키고 다른 노드에서 각 게임 영상을 획득할 수 있는 이중 서버 구조를 제안한다. 몇가지 실험을 실시하여 기술적 가능성을 알아본다.

ABSTRACT

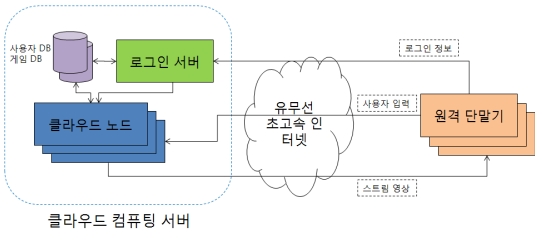
The wide-spread use of the broadband Internet service makes the cloud computing-based gaming service possible. A game program is executed on a cloud node and its live image is fed into a remote user's display device via video streaming. The user's input is immediately transmitted and applied to the game. The minimization of the time to process remote user's input and transmit the live image back to the user and thus satisfying the requirement of instant responsiveness for gaming makes it possible. However, the cost to build its servers can be very expensive to provide high quality 3D games because a general purpose graphics system that cloud nodes are likely to have for the service supports a single 3D application at a time. Thus, the server must have a technology of 'realtime multiple rendering' to execute multiple 3D games simultaneously. This paper proposes a new architecture of 2-tier servers of clouds nodes of which one group executes multiple games and the other produces game's live images. It also performs a few experimentations to prove the feasibility of the new architecture.

Keywords : +cloud computing(클라우드 컴퓨팅), remote gaming(원격 게이밍), live streaming (실시간 스트리밍), multiple rendering(다중 렌더링)

접수일자 : 2012년 04월 02일 심사완료 : 2012년 05월 30일
※ 본 연구는 성공회대학교의 교내 연구비 지원 사업의 일부로 수행된 연구 결과입니다.

1. 서 론

광 대역폭을 지원하는 초고속 인터넷의 일반적인 보급은 IT 기술뿐 아니라 우리의 일상생활을 다양하게 변화시키고 있다. 광 대역폭을 지원하는 인터넷의 등장으로 가능하게 된 기술 중의 하나가 [그림 1]과 같은 클라우드 컴퓨팅 기반 게임 서비스 기술이다. 컴퓨터 게임을 클라우드 노드에서 실행하고 생성된 게임 영상을 영상 스트리밍 기술을 활용하여 원격 단말기에 전송한다. 사용자는 전송된 게임 영상을 보면서 게임을 즐기는 방식이다[1,5,6,7].

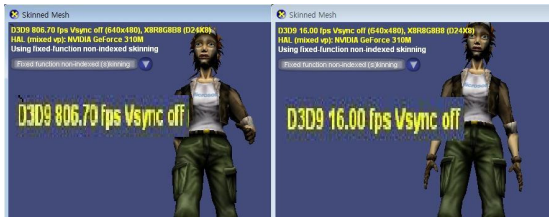


[그림 1] 클라우드 컴퓨팅 기반 게임 서비스

이러한 방식으로 게임을 즐길 수 있다면 사용자는 고가의 게임 하드웨어(게임 콘솔 등)를 별도로 구입하지 않고 게임을 즐길 수 있다. 게임 타이틀을 구입할 필요가 없으며 게임을 즐기기 위한 목적으로 하드웨어를 업그레이드할 필요가 없다. 일정한 월 서비스 이용료를 지불하고 저렴하게 게임을 즐길 수 있기 때문이다[1].

클라우드 컴퓨팅 기술을 활용하여 게임을 사용자에게 서비스하기 위해서는 몇가지 해결해야 하는 기술적 문제가 있다. 그 중의 하나는 고품질 영상에 대한 실시간 인코딩 및 디코딩 기술이다. 게임의 영상이 실시간으로 사용자에게 제공되어야 하기 때문이다. 이러한 기술은 스포츠 중계 등의 실시간 중계 서비스가 일반화되면서 많은 기술적 발전을 이루고 있다[8,9]. 다른 문제는 많은 사용자 입력에 대한 게임의 처리가 즉각적으로 이루어져야 한다는 것이다. 초고속 인터넷의 보급으로 네트워크 지연은 전반적으로 많이 개선되었으나 모든 게임 플레이의 실시간성을 보증하기 위해서는 인터넷 속도가 항상 안정적으로 유지되어야 한다. 또 다른 문제는 서비스에 사용되는 서버를 낮은 비용으로 구축하는 것이다. 서버 구축에 소요되는 비용과 비례할 수밖에 없는 서비스 비용이 사용자가 게임을 기존의 방식으로 즐기는 경우보다 높아진다면 서비스의 상업적 성공은 어려워질 수 있다[1,3].

일반적인 컴퓨터가 지원하는 그래픽 시스템은 한 번에 하나의 그래픽 어플리케이션을 지원하도록 설계되어 있다. 다수의 그래픽 하드웨어를 탑재한 시스템도 여러 개의 하드웨어를 통합하는 하나의 가상 하드웨어를 토대로 그래픽 시스템을 구현하는 방식이다[4]. 한 번에 하나의 그래픽 어플리케이션을 지원하는 그래픽 시스템의 설계는 동시에 2개 이상의 어플리케이션을 효율적으로 지원할 수 없다. 이러한 문제는 [그림 2]와 같이 간단한 3D 어플리케이션을 실행하는 경우 명백해진다. 똑 같은 어플리케이션을 동시에 실행하는 경우 전면(foreground)에 있는 어플리케이션의 프레임 수는 초당 800이상이고 후면(background)에 있는 어플리케이션의 프레임 수는 초당 16 정도이다(실행된 어플리케이션은 DirectX SDK에 포함된 샘플 어플리케이션임). 이러한 방식의 그래픽 시스템을 활용하여 게임 서비스를 진행한다면, 각 노드에서 한 번에 하나의 3D 게임 밖에 서비스할 수 없다. 저렴한 비용으로 하나의 노드를 구성한다고 하더라도, 하나의 노드에서 다수의 사용자를 서비스할 수 없기 때문에 대규모의 사용자를 서비스해야 하는 경우에는 서버를 구축하기 위한 비용이 증가하고 이는 서비스 이용료의 증가로 이어질 수 있다.



[그림 2] 3D 어플리케이션의 전경 및 배경 실행의 예

저렴하게 서버를 구축하기 위해서는 하나의 노드에서 다수의 게임을 동시에 실행하고 영상을 생성, 제공하는 기술이 필요하다. 이러한 기술의 구현을 위해서 본 논문은 하나의 노도에서 다수의 게임 영상을 동시에 생성할 수 있는 실시간 다중 렌더링 기술을 제안한다. 특히, 이러한 기술을 실현하기 위하여 필요한 이중 서버 구조를 제안한다. 제안된 구조의 실현 가능성을 확인하기 위하여 몇가지 실험을 실시하고 그 결과를 통하여 제안된 구조의 기술적 가능성을 입증한다. 2장은 클라이언트-서버 기반의 실시간 다중 렌더링 기술을 구현하기 위하여 필요한 기존 기술을 알아본다. 3장은 실시간 다중 렌더링을 위한 이중 서버 구조를 제안한다. 4장은 제안된 구조가 기술적으로 실현 가능한 지 여부를 확인하기 위하여 몇가지 실험을 시행하고 그 결과를 알아본다. 5장에서 결론을 맺는다.

2. 배경 기술

2.1 클라우드 컴퓨팅 기반 게임 서비스

클라우드 컴퓨팅 기술을 이용한 게임 서비스는 컴퓨터 게임 분야에서 새로운 게임 서비스 방식으로 많은 관심을 받고 있다. [그림 2]와 같이 클라우드를 구성하는 노드에 사용자가 원하는 게임을 실행하고 생성된 게임 영상을 영상 스트리밍 기술을 활용하여 사용자의 단말기에 전달한다. 사용자는 단말기의 영상을 참조하여 게임을 조작한다. 사용자 입력은 유무선 네트워크를 통하여 클라우드 노드에 전달되고 실행 중인 게임에 적용된다.

컴퓨터 게임은 사용자 입력에 대한 실시간적인 적용, 다시 말해 즉각적인 반응성이 게임 플레이 측면에서 중요하다. 네트워크의 속도가 느린 경우에는 실현 불가능한 기술이다. 최근에 광범위하게 보급되고 있는 초고속 인터넷은 14.4Mbps 이상의 속도가 일반적이며[1,2] 이러한 속도는 게임의 즉각적인 반응성을 만족한다[5]. 네트워크의 속도 뿐 아니라, 네트워크 속도의 안정적인 유지도 중요하다. 평균적인 속도가 즉각적인 반응성을 가능하게 하더라고 반응성이 떨어지는 게임 장면이 있다면 게임 플레이에 대하여 사용자의 만족도는 떨어지기 때문이다.

대표적인 서비스로서 온라이브(OnLive), 가이카이(GaiKai), 게임즈앳라지(Games@Large) 등이 있다[5,6,7]. 선도 업체인 미국의 온라이브사는 2010년 6월 자국 내에서 상업 서비스를 시작하였고 2011년 9월에는 영국에서도 서비스를 시작하였다[5].

게임 개발사 또는 제작사가 새로운 플랫폼에서의 서비스를 위하여 게임을 새롭게 개발하거나 변경하는 것은 쉽지 않다. 추가적인 개발 비용이나 라이센스 비용이 부분적인 이유이다. 이러한 이유로 클라우드 컴퓨팅 기반 게임 서비스를 시행하기 위해서는 기 개발된 게임 어플리케이션을 수정 없이 서비스할 수 있는 기술이 필요하다.

2.2 실시간 원격 렌더링 기술

게임즈앳라지는 클라우드 컴퓨팅 기반 게임 서비스 기술로 2가지를 제안하고 있다[10,11,12,13]. 하나는 타 기술처럼 동영상 스트림을 제공하는 동영상 기반 서비스 기술이고, 다른 하나는 그래픽 API 함수를 원격 단말기에서 실행하는 기술이다. 후자의 기술을 적용하기 위해서는 원격 단말기가 3D 그래픽 API를 실시간에 수행하기 위한 그래픽 하드웨어를 보유하고 있어야 한다. 전자의 기술은 사용자 단말기가 3D 그래픽 하드웨어를 보유하고 있지 않으나 동영상 스트림을 실시간으로 재생할 수 있는 경우에 적용 가능한 기술이다. 게임즈앳라지는 가장 일반적으로 사용되는 그래픽 API 중 하나인 OpenGL을 지원하는 원격 렌더링 기술을 구현하였다. 게임즈앳라지의 원격 렌더링 기술은 하나의 어플리케이션이 호출하는 그래픽 함수를 가로채어 원격 단말기에 전송하고 실행하는 방법으로, 다수의 어플리케이션에 대한 렌더링은 지원하지 않는다.

실시간 원격 렌더링 방법이 최초로 시도된 기술 중의 하나가 X 윈도우 시스템이다[14,15]. 1984년 미국 MIT에서 개발된 소프트웨어 시스템과 네트

워크 프로토콜로서, 클라이언트에서 실행되는 그래픽 어플리케이션이 서버에서 표시되는 시스템이며, 그래픽의 표현에 제한적인 기능을 보유한 클라이언트를 대형 모니터를 갖춘 고성능의 서버가 지원하는 방법이다. 클라이언트의 그래픽 영상 구성에 필요한 GLX, OpenGL의 API 함수를 서버에 전송하고 서버에서 실행한다[14,16,18]. 다수의 어플리케이션을 하나의 서버에서 렌더링할 수 있지만, 게임 개발 API로 널리 사용 중인 DirectX나 OpenGL를 사용하여 단일 시스템에서 실행하는 게임을 게임 어플리케이션에 대한 수정 없이 서비스할 수 없다. 또한, X 윈도우 시스템은 사용자 단말기가 고성능 그래픽 하드웨어와 대형 모니터로 구성된 경우를 대상으로 일반적으로 운용된다. 시스템의 네트워크 성능을 높이기 위하여 네트워크 패킷 데이터를 압축할 수 있다[17].

2.3 API 후킹(Hooking) 기술

어플리케이션이 호출하는 그래픽 함수를 가로채어 렌더링 서버에 전송하기 위해서 API 후킹 기술이 필요하다. API 후킹 방식은 크게 물리적 변경 방식과 실시간 변경 방식이 있다[19,20]. 물리적 변경 방식은 실행 파일이나 라이브러리가 실행되기 전에 수정하는 방식으로 크게 3가지 방법이 있다. 첫째는 함수의 시작 위치의 코드를 변경하여 타 라이브러리를 동적으로 로드하고 로딩된 라이브러리의 함수를 호출하는 방법이다. 둘째는 실행 파일의 임포트 어드레스 테이블(import address table)을 수정하여 같은 효과를 얻는 방법이다. 첫 번째 방법은 실행되는 함수를 변경하지만 두 번째 방법은 실행 파일을 직접 수정하는 방법이다. 세 번째는 호출되는 API 함수가 포함되어 있는 라이브러리에 대한 대체 라이브러리 또는 래퍼(wrapper) 라이브러리를 개발하여 활용하는 방법이다. 원 라이브러리와 같은 함수를 포함하지만 각 함수는 다른 기능을 수행한다. 원 라이브러리의 함수가 호출되는 경우 래퍼 라이브러리의 함수가 대신 작동한다. 래퍼 라이브러리의 함수는 변경된 작업을 수행하거나 추가적으로 원 라이브러리 함수를 호출할 수 있다. 원래 호출되는 함수나 로딩되는 라이브러리를 수정하지 않고 함수를 가로챌 수 있으며, 실행 파일에 대한 수정도 필요 없는 방법이다. 클라우드 기반 게임 서비스는 실행 파일을 수정하지 않고 서비스해야 하는 제약이 있기 때문에 함수 시작 위치의 코드를 변경하거나 임포트 어드레스 테이블을 수정할 수 없는 경우가 있다.

실시간 변경 방식은 운영체제나 어플리케이션이 이벤트 훅(hook)을 설정할 수 있는 허가가 있는 경우에 가능한 방법이다. 키보드, 마우스 이벤트에 대한 추가, 삭제, 처리, 변경이 가능하고 네트워크에 대한 이벤트 훅이 가능한 경우도 있다. 만약 그런 허가가 없다면 함수의 시작 코드를 수정하여 새로 삽입된 함수가 대신 실행되도록 할 수 있다. 공유 라이브러리를 지원하는 시스템에서는 메모리 영역에 있는 인터럽트 벡터 테이블이나 임포트 디스크립션 테이블을 수정할 수 있다.

2.4 실시간 다중 렌더링 기술

기존의 다중 렌더링 기술은 고품질 영상을 생성하기 위하여 다수의 독립적인 렌더링 알고리즘을 순차적으로 적용하는 멀티 패스(multi-pass) 렌더링기술이다. 획득하고자 하는 영상의 품질과 생성 목적에 따라 실시간으로 또는 비 실시간으로 이루어진다. 이러한 기술은 다수의 3D 어플리케이션을 동시에 실행시키면서 렌더링을 실시간에 처리해야 하는 경우에 적용할 수 없다. 통상적으로 하나의 장면(scene) 데이터에 다수의 렌더링 기법을 순차적으로 적용시킬 수 있지만, 다수의 3D 어플리케이션이 갖는 전혀 다른 장면 데이터에 적용할 수 없다. 장면 데이터의 용량이 과도하게 많아지거나 매번 비디오 메모리의 데이터를 교체해야 하기 때문에 시스템적 부하가 커진다. 비 실시간적 방법의 대표적인 예는 픽사(Pixar)사의 렌더맨(RenderMan), 엔비디아(Nvidia)사의 멘탈레이(mental ray)이고, 실시간 다중 패스 렌더링은 고품질 게임 영상의 표현을 위하여 대부분의 게임

타이틀 및 게임 엔진에서 사용되고 있다.

다른 다중 렌더링 방식은 다수의 부속 창(sub-window)을 이용하여 장면을 표현하는 방법이다. 하나의 장면 데이터를 다양한 카메라 시점에서 표현할 수 있고, 실제로 마야(Maya)나 맥스(3ds max) 같은 3D 영상 제작 툴에서 사용하고 있다[21]. 이러한 방식을 다수의 어플리케이션에 대한 다중 렌더링에 적용하려면 대규모 장면 데이터를 효과적으로 처리해야하며, 따라서 그래픽 자원의 공유를 효과적으로 처리하기 위한 시스템적 지원이 필요하다.

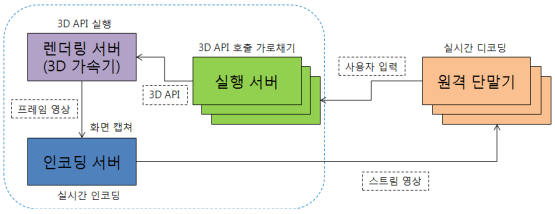
또 다른 방식은 고성능 그래픽 시스템의 자원을 시간 공유(Time-Sharing) 방식으로 여러 3D 어플리케이션이 사용하는 방식이다. CPU를 활용한 병렬 처리 방식과 유사한 방법으로 현재 마이크로소프트사에서 단계적으로 개발 중에 있다[22]. 윈도우 비스타와 함께 개발된 DirectX 10은 WDM(Windows Display Model)을 개발하여 다중 쓰레드에 의한 렌더링 기술을 지원하고 있다.

3. 다중 렌더링 용 이중 서버 구조

3.1 개괄적 구조

[그림 3]처럼 하나 이상의 3D 어플리케이션을 실행하는 어플리케이션 서버와 다수의 어플리케이션에 대한 렌더링을 수행하는 렌더링 서버로 구성한다. 렌더링 서버에서 어플리케이션의 실행을 동시에 수행할 수 있으나 어플리케이션을 별도의 시스템에서 실행한다면 시스템 부하를 분산시킬 수 있다. 렌더링 서버는 영상 캡쳐를 수행하거나 추가적으로 영상 스트림 인코딩의 기능을 담당할 수 있다.

어플리케이션 서버에서 API 후킹 모듈이 구현되어 어플리케이션이 호출하는 그래픽 API 함수를 가로채어 렌더링 서버에 전송한다. 함수 전송은 어플리케이션 서버에서 구현되는 API 함수 전송 모듈이 담당한다. 렌더링 서버는 여러 어플리케이션에서 전송되어온 API 함수 호출을 저장하고 렌더링하는 통합 렌더링 모듈로 구성되어 있다.



[그림 3] 실시간 다중 렌더링을 위한 서버 구조

어플리케이션 서버와 물리적으로 분리되어 있는 렌더링 서버에서 렌더링이 이루어지기 때문에 어플리케이션이 호출하는 함수가 받아들이는 인자의 종류에 한계가 있다. 구체적으로, 실행 중인 어플리케이션의 메모리 영역을 참조할 수 없다. 또한, 어플리케이션 서버에 탑재된 입출력 장치나 보조기억장치에 접근할 수 없다. 렌더 상태(render state) 변수를 참조해야 하는 경우에는 렌더링 서버에서 참조에 대한 요청을 받고 변수에 대한 현재 상태를 반환하기 위해서 네트워크 패킷을 주고받아야 한다.

본 논문에서 구성하고 실험하는 구조는 OpenGL 그래픽 라이브러리를 기준으로 한다. OpenGL은 많은 3D 어플리케이션 개발에 사용되고 있고 많은 종류의 플랫폼에서 지원되고 있다. 게임 개발용으로 널리 사용되는 DirectX를 지원하기 위해서 변경할 수 있다.

3.2 API 후킹(Hooking) 모듈

API 후킹 모듈은 클라이언트에서 3D 어플리케이션이 실행되면 당 어플리케이션에서 호출하는 그래픽 라이브러리의 함수를 가로채는 역할을 수행한다. API 함수는 API 함수 전송 모듈을 통하여 렌더링 서버에서 동작하는 통합 렌더링 모듈에 전송된다.

기존 3D 어플리케이션을 수정하지 않고 서비스해야하기 때문에, 기존 API 함수를 변경하지 않고

동적 라이브러리의 래퍼 라이브러리 또는 프락시(proxy) 라이브러리를 구현하여 후킹을 수행한다. OpenGL 기반의 3D 어플리케이션 개발에 사용되는 동적 라이브러리인 glut, glu, opengl에 대한 래퍼 라이브러리를 구현한다. 본 연구에서 사용하는 라이브러리들은 OpenGL 1.1에 기반하며, 각각 119개, 52개, 368개의 함수를 포함하고 있다([표 1] 참조). 각 래퍼 라이브러리는 이들에 대한 함수를 모두 구현한다. 어플리케이션이 OpenGL 함수를 호출하면 래퍼 라이브러리의 함수가 대신 호출되면서 호출된 내용이 래퍼 라이브러리의 일부로 구현되는 API 함수 전송 모듈에 의해서 통합 렌더링 모듈에 전송된다. 각 래퍼 라이브러리는 라이브러리가 로딩되면서 원래의 라이브러리를 로딩한다. 이렇게 구현된다면 어플리케이션이 어플리케이션 서버에서 함수를 실행하여 렌더링을 수행할 수 있는 장점이 있다.

3.3 API 함수 전송 모듈

각 API 함수는 함수가 호출되는 즉시 서버에 전송되는 방식을 기본으로 한다. 이러한 방식은 각 어플리케이션이 수행하는 렌더링 루프에 해당하는 함수의 개수가 증가할수록 전체 함수 전달 시간이 비례적으로 증가하는 특성이 있다. 복잡한 어플리케이션을 처리하기 위해서는 여러 개의 함수 호출을 통합하여 보내는 방법이 필요하다. 시스템 및 서비스 환경에 따라서 한 번에 보낼 수 있는 함수의 개수를 조절할 수 있도록 변경 가능한 패킷을 설계하여 적용한다.

그래픽 라이브러리 함수는 [표 1]과 같이 50% 이상이 대부분 값 호출(call-by-value) 형태이기 때문에 가로챈 함수를 렌더링 모듈에 전송하고 추가적인 처리 없이 실행할 수 있다. 하지만 함수가 실행 프로그램의 메모리 영역을 주소 값으로 참조하거나 입출력장치나 보조기억장치의 접근이 필요한 경우라면 별도의 처리 과정이 요구된다. 참조 호출인 경우라도 즉각적으로 대체할 수 있는 값 호출 형태의 함수를 대신 사용할 수 있기 때문에 메모리 영역을 참조하는 함수의 개수는 줄어든다(대체 방법이나 구분되는 함수의 개수는 개발 환경과 목적에 따라 달라질 수 있음).

[표 1] OpenGL API 함수의 호출 방식

		gl	glu	glut	구성비
값 호출		183	6	86	51.0
참조 호출	대체 함수	96	0	0	17.8
	수정 대체	33	41	9	15.4
	별도 처리	24	4	1	5.4
내부함수		7	1	6	2.6
MS 윈도우즈		25	0	0	4.6
사용자 입력		0	0	17	3.2
합 계		368	52	119	100

대표적인 예가 void glVertex3fv(const GLfloat *v)이다. 3개의 실수 값을 배열 형태로 저장하고 배열의 주소 값을 인자로 전달하는 이 함수는 void glVertex3f (GLfloat x, GLfloat y, GLfloat z)를 대신 호출하고 3개의 실수 값을 인자로 전달하는 것으로 대체할 수 있다. 대규모의 메모리의 영역을 참조하거나 보조기억장치의 파일을 참조해야하는 함수의 비율은 5% 정도를 차지한다. 이러한 함수들의 특징은 렌더링 루프에서 매번 호출하는 함수가 아니고 렌더링 루프가 시작하기 전에 호출되거나 별도의 이벤트에 의해 한번 호출되는 함수들이다. 별도의 처리를 위하여 렌더링 서버에서 추가적인 기능을 구현할 수 있다.

함수 패킷의 전송을 위해서 프로세서 간 통신 수단인 파이프(pipe)를 활용한다. 파이프는 로컬 컴퓨터에서 사용되는 무명 파이프와 서버와 다수의 클라이언트 사이에 사용되는 지정 파이프가 있다[23]. 본 연구에서는 렌더링 서버에 파이프 서버를 구현하고 어플리케이션 서버에 파이프 클라이언트 부분을 구현하여 사용한다. 지정 파이프는 하나의 파이프 서버에 다수의 클라이언트가 통신할 수 있고 또한, 다수의 어플리케이션 서버에서 동작하는 클라이언트와의 통신이 가능하기 때문에 하나의 렌더링 서버가 다수의 어플리케이션 서버를 지원할 수 있도록 유동적으로 설정할 수 있는 장점이 있

다. 지정 파이프는 양방향 통신을 지원하도록 설정하고 렌더링 모듈에서 수행된 함수 호출에 대한 결과를 어플리케이션에 전송한다.

3.4 통합 렌더링 모듈

렌더링 서버의 통합 렌더링 모듈은 여러 어플리케이션에서 전송되어온 API 함수를 취합하고 각 어플리케이션의 현재 프레임 영상을 당 서버에 탑재되어 있는 그래픽 시스템을 활용하여 생성한다. 본 모듈은 렌더링 서버의 출력 가능 영역을 어플리케이션 별로 구분하고 나누어진 영역에 각 어플리케이션의 영상을 생성한다. 렌더링 모듈에 관리하는 부속 윈도우(sub-window)를 사용하거나 하나의 윈도우를 여러 개의 뷰포트(viewport)로 구분하여 사용한다.

어플리케이션은 자체적으로 필요한 렌더 스테이트(render state)를 설정하고 렌더링을 수행하기 때문에, 본 연구의 렌더링 모듈이 렌더링하는 각 어플리케이션은 렌더 스테이트가 서로 다르다. 이에 대한 처리를 위하여, 각 어플리케이션이 호출하는 렌더 스테이트 설정 관련 함수 호출을 저장하고, 각 어플리케이션의 렌더 루프에 대한 처리를 시작하기 직전에 저장된 함수를 호출하여 각 어플리케이션이 필요한 렌더 스테이트를 설정하는 방식을 채택한다. 렌더링 모듈의 렌더링 루프는 각 어플리케이션에 대한 렌더링을 순차적으로 수행한다. 따라서 렌더링 루프는 각 어플리케이션의 렌더링을 시작하기 전에 렌더 스테이트를 변경한다. 통상적으로 렌더 스테이트를 변경하는 작업은 시스템적 오버헤드를 발생시켜 렌더링 시간이 원 어플리케이션의 총 렌더링시간보다 많아진다. 렌더 스테이트를 매번 변경할 뿐 아니라 그래픽 파이프라인에서 빈번하게 사용되는 행렬(월드 행렬, 뷰 행렬, 투영 행렬 등)을 바꾸어줘야 한다.

4. 실 험

본 논문에서 제안하고 있는 실시간 원격 다중 렌더링 시스템의 기술적 가능성을 검증하기 위하여 여러 가지 실험을 수행한다. 특히, 전송된 API 함수에 대한 통합 렌더링 모듈에서의 렌더링 시간과 API 패킷의 전송시간을 확인할 수 있도록 실험을 수행한다. 본 실험에서는 3D 그래픽 프로그램 샘플 사이트인 www.codesampler.com의 4개의 샘플 프로그램을 변형하여 사용하며, 각 샘플이 호출하는 API 함수의 개수는 [표 2]와 같이 구성되어 있다.

[표 2] 샘플 프로그램의 구성

	총API	glut	glu	gl	셋업	루프
샘플 1	22	10	0	12	12	10
샘플 2	29	11	2	16	17	12
샘플 3	42	14	2	26	14	28
샘플 4	3237	10	1	3226	27	3210

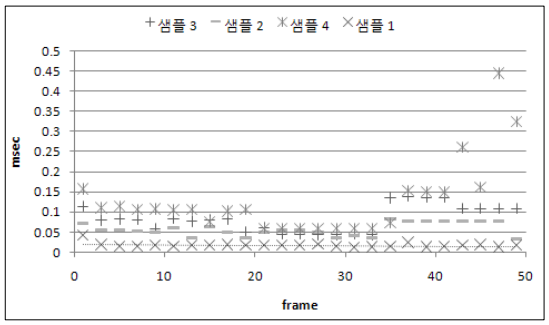
샘플 1은 간단한 2D 사각형을 렌더링하는 기본적인 프로그램이다. 샘플 2는 glutWireSphere() 함수를 이용하여 구를 와이어프레임 형태로 렌더링하는 프로그램이며 내부적으로 전후면 클리핑을 수행한다. 샘플 3은 glutSolidSphere()를 추가적으로 활용하여 세 개의 구가 태양, 지구, 달의 움직임을 표현하는 프로그램이다. 샘플 4는 지구를 표현하는 구에 텍스쳐와 광원 효과를 적용하여 실제 지구와 같은 모습을 표현하는 프로그램이다. 샘플 4에서는 glutWireSphere()나 glutSolidSphere()를 사용하지 않고 구를 표현하기 때문에 많은 gl함수를 호출하고 있다. 샘플 2와 3에서 호출하는 glutWireSphere()나 glutSolidSphere()는 본 함수들이 궁극적으로 gl함수를 호출하기 때문에 실제로 본 실험에서 전송되는 API 함수는 gl 함수이다. 실제 서비스 시스템에서는 패킷 전송에 대한 부하를 감소시키기 위해서 glutWireSphere()나 glutSolidSphere()와 같이 glu나 glut가 제공하는 복잡한 객체 생성 함수를 직접 전송하는 것이 바람직하다.

본 실험에 사용된 컴퓨터는 Intel core i5 760(2.80Ghz) CPU를 채택하고 있다. 운영체제로 MS Windows 7 Home Premium K(32bit)를 탑

재하고 있으며 4.0GB의 주 메모리를 보유하고 있다. 본 실험은 어플리케이션 서버와 렌더링 서버를 하나의 물리적 시스템에서 구현하여 수행한다.

4.1 렌더링 시간 측정

앞에서 언급된 것처럼 원격 렌더링 시스템은 로컬 렌더링에 비해서 많은 시간이 소요된다. 로컬 렌더링 시 소요되는 평균적인 시간은 샘플 1이 0.02msec, 샘플 2가 0.06msec, 샘플 3이 0.09msec, 샘플 4가 0.14msec가 소요된다([그림 4]참조).



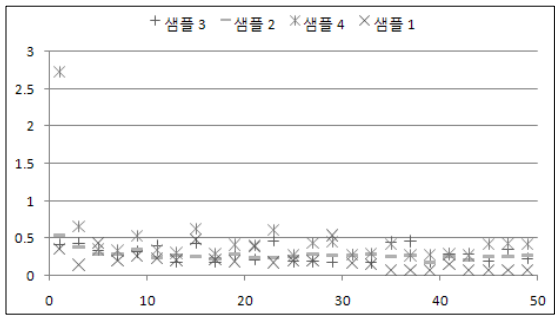
[그림 4] 독립적 실행 시 샘플의 렌더링 시간
(홀수 번째 프레임의 시간 표시)

다수의 3D 어플리케이션을 위해서 원격 렌더링을 수행하는 경우, 렌더링 상태를 어플리케이션에 따라서 변경해줘야 하기 때문에 그에 대한 부하가 발생하여 렌더링 시간이 추가적으로 소요된다. 원격 렌더링 시 소요되는 평균적인 시간은 샘플 1이 0.07msec, 샘플 2가 0.54msec, 샘플 3이 0.91msec, 샘플 4가 6.87msec가 소요된다([그림 5]참조).

이는 각각 3.5배, 9배, 10.1배, 49배가 증가한 수치이다. 이러한 수치는 전체 윈도우의 프레임 영상을 생성하는데 많은 시간이 소요될 수 있음을 의미한다. 하지만, 전체 영상의 생성에 9.04msec(110.62fps에 해당)가 소요되기 때문에 서비스하는 어플리케이션의 종류와 그 수에 따라서 서비스가 가능할 수 있음을 의미한다.

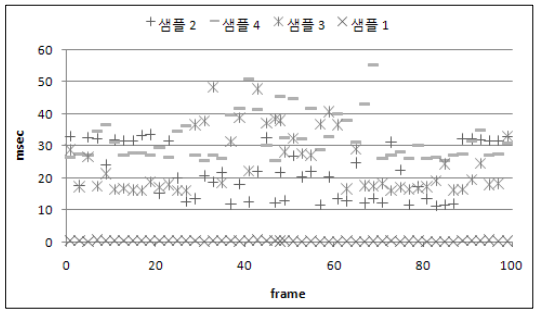
4.2 패킷 전송 시간 측정

실시간 원격 다중 렌더링 시스템이 실시간 렌더링이라는 목적을 달성하기 위해서는 렌더링 속도 이외에도 패킷 전송 속도가 빨라야 한다.



[그림 5] 통합 렌더링 모듈에서 각 샘플의 렌더링 시간(홀수 번째 프레임의 시간 표시)

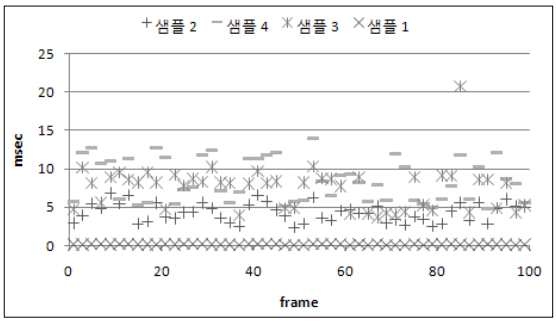
하지만, 본 연구에서 구현한 파이프에 의한 API 함수 패킷의 전송 방식은 많은 시간이 소요된다. 샘플 1의 경우 0.28msec, 샘플 2의 경우 20.94msec, 샘플 3의 경우 24.66msec, 샘플 4의 경우 31.9msec가 소요된다([그림 6] 참조). 간단한 샘플임에도 4개의 어플리케이션에 필요한 패킷의 전달에 77.78msec (12.86fps에 해당)가 소요되어 게임 서비스에서 요구되는 실시간 처리가 불가능하다.



[그림 6] 각 샘플의 패킷 전송 시간
(홀수 번째 프레임의 시간 표시)

타 어플리케이션이 렌더링되는 도중에 패킷을 전달할 수 있기 때문에 실현할 수 있는 프레임 레이트는 그 이상이 될 수 있다. 더욱이 API 함수의

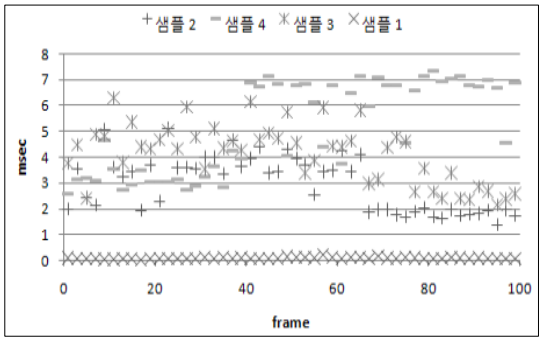
호출이 전송과 즉시 이루어지지 않는 특성을 이용하여, 클라이언트에서 API 호출을 통합하여 전송함으로써 전송속도를 개선할 수 있다. 본 연구에서 128바이트의 최대 패킷 크기를 적용하는 경우, 샘플 1의 경우 0.1msec, 샘플 2의 경우 4.4msec, 샘플 3의 경우 7.07msec, 샘플 4의 경우 8.53msec가 소요되어 종합적으로 20.1msec (49.75fps)가 소요되는 개선을 이룰 수 있었다([그림 7] 참조).



[그림 7] 최대 128바이트의 패킷 크기를 허용하는 경우 전송 시간(홀수 번째 프레임의 시간 표시)

최대 패킷 크기를 256 바이트와 384 바이트로 확대하는 경우 추가적인 전송속도의 개선이 있다([그림 8,9] 참조). 각각 12.4msec와 9.8msec의 전송 시간이 필요하고, 이는 80.6fps와 120.0fps의 프레임 레이트에 해당된다.

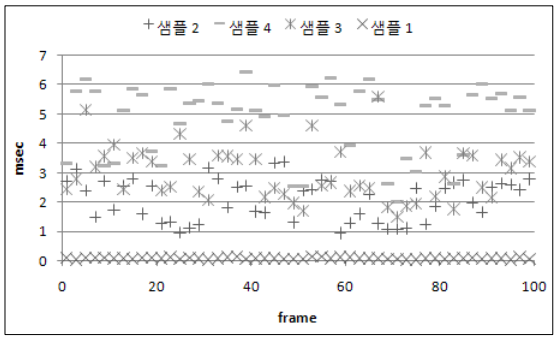
[그림 10]은 서버에서 동작하는 원격 다중 렌더링 시스템의 실행 예이다.



[그림 8] 최대 256 바이트의 패킷 크기를 허용하는 경우의 전송 시간(홀수 번째 프레임의 시간 표시)

5. 결 론

4장의 실험에서 확인된 바와 같이 실시간 원격 다중 렌더링 시스템은 총 렌더링 시간과 API 패킷 전송 시간을 고려할 때, 각 3D 어플리케이션이 갖는 장면의 복잡성에 좌우되지만 많은 어플리케이션의 서비스가 가능한 것으로 확인된다.



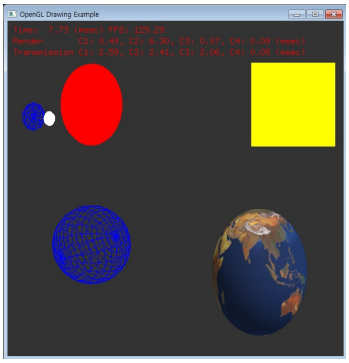
[그림 9] 최대 384 바이트의 패킷 크기를 허용하는 경우의 전송 시간(홀수 번째 프레임의 시간 표시)

고성능의 GPU가 저렴하게 공급되고 있기 때문에 본 논문에서 제안하는 구조를 적용하여 고성능 GPU를 사용한다면 경제적으로 렌더링 서버를 구축할 수 있다. 본 구조는 서버의 구축에 필요한 비용을 절감시킬 뿐 아니라, 서버의 시스템적 부담을 효과적으로 분산하여 더 나은 서비스를 제공할 수 있다.

본 연구에서 제안된 구조의 장점은 다음과 같이 요약할 수 있다.

1. 하나의 그래픽 시스템에서 다수의 3D 게임을 실행함으로써 클라우드 기반 게임 서비스에 필요한 서버를 저렴한 비용으로 구축할 수 있다.
2. 어플리케이션 서버와 렌더링 서버를 분리함으로써 시스템 부하를 분산시킬 수 있고 서비스 환경에 따라 유동적으로 서비스 시스템을 구성할 수 있다.
3. 게임 타이틀에 대한 수정 없이 게임 서비스를 제공할 수 있다.

향후 연구에서는 OpenGL 라이브러리를 활용한 게임에 적용하여 제안된 기술을 개선하고 상업적 가능성을 실증할 예정이다. 또한, 네트워크 패킷 전송 속도 및 렌더링 속도를 개선하기 위한 연구를 진행할 예정이다. 또한, DirectX를 이용한 게임을 지원하기 위해서 당 그래픽 라이브러리를 이용한 실시간 원격 다중 렌더링 기술을 개발할 예정이다.



[그림 10] 원격 다중 렌더링 시스템 실행 예

참고문헌

[1] 임충규, 김성수, 김경일, 원종호, 박창준, “클라우드 컴퓨팅 기반의 게임 스트리밍 기술 동향”, 전자통신동향분석, 26권, 1호, pp47-56, 2011.

[2] OECD, OECD Broadband Statistics, www.oecd.org, 2010.

[3] 임충규, “3D 어플리케이션 스트리밍 서비스를 위한 GPU 공유 방법”, HCI 2011 학술대회 논문집, 2011.

[4] G. Torres, “SLI vs, CrossFile”, http://www.hardwaresecrets.com/article/391, 2008

[5] www.onlive.com.

[6] www.gaikai.com

[7] www.gamesatlarge.edu

[8] 임충규, “모바일 기반 시연 시스템 ‘스마트 데모’의 설계 및 구현”, 한국콘텐츠학회논문지, 11권, 12호, pp1-11, 2011.

[9] http://www.iupui.edu/~nmstream/live/findings.php.

[10] Y. Tzruya, A. Shani, F. Bellotti, A. Jurgelionis, “Games@Large-a new platform for ubiquitous gaming and multimedia”, Broadband Europe Conference 2006, 2006.

[11] P. Eisert, P. Fechteler, "Remote Rendering of Computer Games", SIGMAP 2007, 2007

[12] P. Eisert, P. Fechteler, "Low Delay Streaming of Computer Graphics", International Conference of Image Processing(ICIP), 2009.

[13] A. Jurgelionis, P. Fetchteler, P. Eisert, F. Belloti, H. David, J.P. Laulajainen, R. Carmichael, V. Poulopoulos, A. Laikari, P. Perala, A. De Gloria, C. Bouras, “Platform for Distributed 3D Gaming”, International Journal of Computer Games Technology 2009, 2009.

[14] www.x.org

[15] http://en.wikipedia.org/wiki/X_window_system

[16] www.opengl.org

[17] X Window System Network Performance, http://keithp.com/~keithp/talks/usenix2003/html/net.html

[18] X Window System's Programmer's Guide, http://lesstif.sourceforge.net/doc/super-ux/g1ae04e/contents.html

[19] http://en.wikipedia.org/wiki/Hooking

[20] “Practical Guides on Win32 Hacking and Windows Hacking - Part I”, http://www.scribd.com/

[21] http://usa.autodesk.com/

[22] “Windows Driver Model(WDM)”, http://msdn.microsoft.com/en-us/windows/hardware/gg463453

[23] “Named Pipes”, http://msdn.microsoft.com/en-us/library/windows/desktop/aa365590(v=vs.85).aspx

임 충 규 (Lim, Choong-Gyoo)

1999년 2월-2011년 2월 한국전자통신연구원(ETRI) 책임연구원

2011년 3월-현재 성공회대학교 멀티미디어시스템공학과 교수

관심분야 : 기하 모델링, 컴퓨터 그래픽스, 컴퓨터 게임