Journal of Korea Game Society 2014 Oct; 14(5): 117-126
http://dx.doi.org/10.7583/JKGS.2014.14.5.117

게임 프로그래밍

Direct3D API의 원격 실시간 실행 시스템 개발

임충규
성공회대학교 컴퓨터공학과
cglim@skhu.ac.kr

Development of a Remote Rendering System using Direct3D API

Choong-Gyoo Lim
Dept. of Computer Science and Engineering, SungKongHoe University

요 약

레가시 3D API를 이용한 원격 렌더링 시스템을 개발한다면 다양한 응용 분야가 있다. 실시간 비디오 스트리밍 기반 클라우드 게이밍 서비스의 구현에 활용하거나, 다수의 3D 어플리케이션에 대한 렌더링을 지원하는 GPU 가상화의 구현 등에 활용할 수 있다. OpenGL API은 독립적인 전역함수로 구성되어 있고, Direct3D API는 마이크로소프트의 COM 기술 기반의 인터페이스와 그 멤버함수로 구성되어 있다. 본 논문은 상대적으로 구현이 복잡한 Direct3D에 대한 원격 렌더링 시스템을 성공적으로 설계하고 구현함으로써 일반적인 레가시 3D API에 대한 적용 가능성을 확인하고자 한다. 본 연구에서 구현한 원격 렌더링 시스템을 샘플 Direct3D 어플리케이션에 적용하고, 몇가지 실험을 실시하여 기술적 가능성을 확인한다.

ABSTRACT

There are various kinds of applications if one can develop a remote execution system using for legacy 3D APIs. It can be used in implementing a cloud gaming service based on the real-time video streaming technology. Or, it can also be used in implementing a GPU virtualization for simultaneously rendering of many different 3D applications. The OpenGL API consists of independent global functions while the Direct3D API consists of Microsoft COM-based interfaces and their member functions, which makes the implementation of remote rendering system more difficult. The purpose of the paper is to show the applicability of the technology to any legacy 3D API by successfully designing and implementing a remote rendering system using the Direct3D API. It applies the implementation to a sample Direct3D application and also performs a few experimentations to show the technical feasibility.

Keywords : remote rendering(원격 렌더링), cloud gaming(클라우드 게이밍), GPU virtualization(GPU 가상화)

Received: Aug. 26, 2014 Accepted: Sept. 22, 2014
Corresponding Author: Choong-Gyoo Lim(SungKongHoe Univ.)
E-mail: cglim@skhu.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

클라우드 게이밍 서비스는 통신 네트워크의 발전에 따라 IT 산업계로부터 많은 관심을 받고 있다. 2009년 미국의 온라이브(OnLive)사가 실시간 비디오 스트리밍 기반 클라우드 게이밍 서비스를 발표하였고, 2010년 자국에서 상용 서비스를 시작하였다. 하지만, 당시의 3G 네트워크는 지연시간이 100~200msec 정도이기[1] 때문에 이동통신을 통한 클라우드 게이밍 서비스는 어려웠다. 네트워크 지연 이외에도 추가적인 지연시간이 소요되기 때문이다. 최신 이동통신 기술인 LTE-A는 지연속도를 약 36msec 정도까지 낮추었다[2]. 이러한 지연속도는 실시간 게이밍 서비스를 위한 지연시간인 50msec를[3] 만족할 수 있는 수준으로 클라우드 게이밍이 안정적으로 서비스될 수 있는 기반이 된다.

클라우드 게이밍 서비스의 상업적 가능성을 결정짓는 요인 중의 하나는 클라우드 서버의 구축에 소요되는 비용이다. 서버구축 비용 또는 이러한 비용이 반영되는 서비스 이용 요금이 게임 타이틀을 직접 구매하여 즐기는 방식보다 비싸진다면 상업적 서비스의 성공 가능성은 매우 낮아진다[4,5]. 서비스구축 비용을 줄이기 위한 방법으로 하나의 GPU 서버에서 다수의 3D 어플리케이션의 렌더링 작업을 수행하는 GPU 가상화 기술이 개발되고 있다[6,7]. 이외에도 원격 렌더링 및 다중 렌더링을 위한 이중 서버 구조 등의 대안적인 방법이 모색되고 있다[5]. 이러한 구조는 어플리케이션 구동 서버와 렌더링 서버를 분리함으로써 시스템 부하를 효과적으로 분산시킬 수 있는 장점이 있고, 궁극적으로 서버구축 비용을 절감할 수 있다. OpenGL 라이브러리에 대하여 API 함수 가로채기 기술을 적용하여 그 기술적 가능성이 이미 검증되었다[8]. 본 논문에서는 Direct3D 라이브러리에 API 함수 가로채기 기술을 적용하여 원격 렌더링 시스템을 구성하고 기술적 가능성을 실제로 확인하고자 한다. 독립적인 전역함수로 이루어진 OpenGL 라이브러리는 원격 렌더링 시스템의 구성에 필요한 API 함수 가로채기를 상대적으로 쉽게 구현할 수 있는 반면, 마이크로소프트의 COM 기술을 기반으로 구현된 Direct3D 라이브러리는 COM 객체의 인터페이스와 각 인터페이스의 멤버함수, 그리고 다수의 독립적 전역함수로 구성되어 있기 때문에 API 함수 가로채기의 구현이 상대적으로 복잡하다. 본 논문은 Direct3D를 이용한 API 함수 가로채기와 원격 렌더링 기술을 구현함으로써, 이러한 기술이 특정한 3D API에 국한되지 않고 일반적인 3D API에 적용될 수 있음을 보여주고자 한다.

본 논문은 2장에서 원격 렌더링 시스템 구현에 필요한 요소 기술을 고찰한다. 3장은 Direct3D를 활용한 원격 렌더링 시스템을 제안한다. 4장은 Direct3D 기반의 원격 렌더링 시스템을 구성하기 위한 세부적인 구현 내용을 설명한다. 그리고, 원격 렌더링 시스템을 샘플 Direct3D 어플리케이션에 적용하여 몇 가지 실험을 시행함으로써 기술적 가능성을 확인한다. 5장에서 향후 연구 방향을 모색하고 결론을 맺는다.

2. 배경 기술

2.1 클라우드 게이밍 서비스

미국의 온라이브사는 2009년 3월 클라우드 게이밍 서비스를 '온라이브'라는 이름으로 발표하였다. 온라이브는 2010년 6월 자국에서 상업 서비스를 시작하였고, 2011년 9월에는 영국에서 상업 서비스를 개시하였다. 유·무선 네트워크를 활용하여 씬 클라이언트 PC, TV, 모바일단말기 등을 지원한다. H.264 기반의 자체적인 인코딩 알고리즘을 개발·적용하여 실시간 인코딩, 디코딩이 가능한 비디오 스트리밍 기술을 개발하였으며, 네트워크 최적화 기술을 개발하여 네트워크 지연을 최소화하는 것으로 알려져 있다[9]. 2012년 1월에는 자사의 실시간 스트리밍 기술을 활용하여 윈도우7 기반의 데스크탑 PC 가상화 기술을 발표하였다.

미국의 가이카이(GaiKai)사는 2010년 인터넷 브라우저에서 구동하는 클라우드 게이밍 서비스를 발표하고 2011년 국제적인 서비스를 시작하였다. 인터넷 브라우저에 이미 설치되어 있는 자바(Java)나 어도비 플래시(Adobe Flash) 플러그인을 활용하여 서비스를 구현하였으며, 따라서 별도의 하드웨어나 소프트웨어의 설치가 필요 없다[10]. SCE(Sony Computer Entertainment)는 2012년 가이카이를 인수하고 클라우드 스트리밍 기술을 활용하여 'Remote Play'와 'PlayStation Now'라는 서비스를 2013년과 2014년에 각각 발표하였다[1].

유럽위원회(European Committee)에서 추진하여 개발한 게임즈앳라지(Games@Large)는 FP6(Framework Programme)프로젝트의 일부로 개발되었다. 서버에서 구동하는 게임 영상을 모바일 단말기나 노트북 컴퓨터, 디지털 TV에 원격 전송하여 게임을 즐길 수 있도록 서비스하는 기술이다[11,12,13,14]. 타 기술과 같이 H.264 기반의 실시간 비디오 스트리밍 기술을 적용하여 게임 영상을 원격 단말기에 전송한다. 추가적으로, 3D 가속기를 탑재한 원격 단말기에 3D API 함수를 전송하고 이를 단말기에서 호출하여 프레임 영상을 생성하는 '그래픽 스트리밍' 기술을 개발하여 활용하였다. 이러한 기술은 전송되는 데이터 패킷의 양을 대폭 감소시키기 때문에, 안정적인 네트워크 지연 시간을 확보할 수 있도록 한다.

[Table 1] Comparison of Cloud Gaming Services

Service	Video Codec	Frame Image Transmi-ssion	Service Terminal	In service or not
OnLive	H.264	video streaming	thin client..	commerciali zed in 2010
GaiKai	H.264	video streaming	internet browser	integrated into Sony's gaming service
Games @Large	H.264	video/ graphics streaming	digital TV..	not in service

국내에서는 초고속인터넷 업계와 케이블방송 업계에 의해서 클라우드 컴퓨팅 기반 게이밍 서비스가 이루어지고 있다. LG유플러스는 2012년 NVDIA의 비디오 스트리밍 기술을 활용하여 클라우드 게이밍 서비스 'c-games'를 시작하였다[15]. PC나 게임 콘솔에서 즐기던 게임을 스마트TV나 스마트폰에서 즐길 수 있도록 지원하고 있다. 케이블TV 업체 CJ헬로비전은 'X게임'을 서비스하고 있으며, 초고속인터넷 업체 SK브로드밴드와 KT는 각각 'Btv 클라우드 스트리밍 서비스'와 'Wiz 게임'를 서비스하고 있다[16,17].

2.2 GPU 가상화 기술

클라우드 게이밍 서비스를 경제적으로 실현하려면 하나의 물리적 서버를 활용하여 다수의 게임 어플리케이션을 구동해야 한다. 그렇지 않다면 클라우드 게이밍 서비스의 서버구축 비용이 증가하고 따라서 서비스 비용이 증가한다. 하나의 물리적 서버에 가상화 솔루션을 적용하여 다수의 게스트 OS를 탑재하고 이를 활용하여 다수의 게임 어플리케이션을 구동하는 방법이 시도되고 있다. 게스트 OS나 어플리케이션은 독립적으로 부여된 가상의 자원을 활용하여 연산을 수행한다. GPU를 포함한 대부분의 I/O 장치는 일반적으로 단일 어플리케이션이나 소프트웨어를 지원하도록 설계되었으나 가상화 기술을 적용하여 다수의 어플리케이션 또는 다수의 운영체제가 공유하여 활용할 수 있다. GPU 가상화 기술은 독립적으로 구동되는 게스트 OS에서 3D 어플리케이션을 실행함에 있어 호스트 시스템에서 설치된 GPU H/W를 활용한다. 3D API 원격 실행이 적용되는 중요한 형상 중의 하나이다.

GPU 가상화 기술은 작동 방식에 따라 전용 접근 모델, S/W 가상화, H/W 가상화 등으로 구분할 수 있다[18]. ① 전용 접근 모델은 가상 머신에서 호스트 시스템에 탑재된 GPU 장치를 직접 접

1) RemotePlay는 게임 콘솔 PS4와 휴대용 게임기 PS Vita 사이의 게임 스트리밍 서비스이며, PlayStation Now는 게임 콘솔, 스마트 TV 등을 포함하는 클라우드 게이밍 서비스임

근하는 방식이다. 일반적인 시스템의 GPU 장치를 사용하기 위하여 설치되는 그래픽 드라이버를 가상 머신에서 똑 같이 설치하고, 일반 시스템에서와 동일한 3D 가속 기능과 성능을 발휘하도록 한다. 전용 접근 모델에서는 단일 GPU H/W가 하나의 가상 머신과 어플리케이션을 지원한다. ② S/W 가상화 방식은 가상 머신에 탑재된 그래픽 드라이버가 어플리케이션의 3D API를 가로채어 호스트 시스템의 그래픽 드라이버에 전달하여 수행한다. API 가로채기는 VM(Virtual Machine, 가상 기계)에 설치된 GPU 드라이버를 통해서 이루어진다. API 가로채기가 S/W적으로 이루어지기 때문에 스케일러블(Scalable) 특성을 보유하여 동시에 대규모의 가상 머신과 어플리케이션을 지원할 수 있다. ③ H/W 가상화 방식은 하이퍼바이저(Hypervisor)에 설치되는 관리자 프로그램을 통하여 GPU 장치에 직접적인 접근을 허용하는 방식이다. VM에 전용 그래픽 드라이버를 설치함으로써 어플리케이션은 GPU 장치가 제공하는 모든 기능과 성능을 제공받는다. 단일 GPU 장치가 다수의 가상 머신과 어플리케이션을 지원하기 때문에 전용 접근 모델보다 많은 가상 머신과 어플리케이션을 지원하고 S/W 가상화 모델보다 적은 수의 가상 머신과 어플리케이션을 지원한다.

[Table 2] GPU Virtualization Methods

Method	Features	Architecture
Dedicated access	full, direct access to GPU full W/S performance	H/W driver on a VM 1 GPU : 1 VM
S/W Virtualization	scalability of 50+ low cost, limited support	SVGA driver on a VM H/W driver on a Hypervisor
H/W Vertualization	scalability full, direct access to GPU, full W/S performance, low cost	H/W driver on VM direct access to H/W via GPU manager

이상의 GPU 가상화 방법은 하나의 시스템이 보유한 자원을 하나 이상의 VM이 공유하기 위한 방법으로, 물리적으로 분리되어 있는 원격 서버의 GPU 자원에 대한 공유를 지원하지 않는다.

2.3 API 함수 가로채기 기술

Direct3D API 함수를 원격 서버에서 실행하고자 한다면 로컬 서버에서 호출되는 API 함수에 대한 호출 정보를 원격 서버에 전송해야 한다. 로컬 서버에서 호출 정보를 획득하기 위해서는 API 함수 가로채기 기술이 요구된다. 일반적으로 API 가로채기 기법은 대체함수의 추가 시점에 따라 물리적 변경 방법과 실행 시간(Run-time) 변경 방법이 있다[19]. 물리적 변경 방법은 어플리케이션이 실행되기 전에 호출되는 함수를 물리적으로 변경하든지, IAT(Import Address Table)를 수정하든지, 래퍼(Wrapper) 라이브러리로 시스템 라이브러리 파일을 대체하는 등의 기법이 있다. 어플리케이션 실행 시 타겟 프로세스의 메모리 공간을 직접 수정하는 방법인 실행 시간 변경 방법에는 실시간 이벤트 훅 삽입, 타겟 함수 헤더 수정, IAT 수정 등의 방법이 있다. API 라이브러리를 대체하는 API 함수 가로채기 방법으로 윈도우 플랫폼에서 가능한 방법은 대표적으로 DLL Injection, Proxy DLL(또는 래퍼 라이브러리) 등이 있다.

API 가로채기 방법은 크게 대체함수 추가 기능과 가로채기 기능으로 구성되어 있다[20]. 이러한 기능을 수행하는 구체적인 기법은 각각의 API 가로채기 방법에 따라 [Table 3]과 같이 구분된다.

2.4 실시간 원격 렌더링 기술

실시간 원격 렌더링 기술이 최초로 구현된 사례는 X 윈도우 시스템이다. 1984년 MIT가 개발한 소프트웨어 시스템과 네트워크 프로토콜로서 원격 시스템에서 구동하는 어플리케이션에 대한 3D 렌더링을 로컬 디스플레이 서버가 담당한다. 어플리케이션의 영상 생성에 필요한 GLX, OpenGL의

API 함수를 디스플레이 서버에 전송하고 이를 디스플레이 서버가 실행하여 어플리케이션 영상을 생성한다[21,22].

[Table 3] Methods of API Function Interception

Method	Interception	Implanting	Replacement	Complexity
DLL Injection	target function modification or IAT modification	direct copy into memory space of target process	per library or function	medium
	use Detours[2)]	use Detours library		easy
Proxy DLL	direct call of proxy functions	upon load of application	per library	easy

게임즈앳라지가 실시간 비디오 스트리밍 기술의 대안으로 제안하고 있는 그래픽 스트리밍 기술도 원격 렌더링 기법이다. 서버에서 구동하는 어플리케이션의 3D API 함수를 3D 가속기 H/W를 탑재하고 있는 원격 단말기에 전송하고 호출하는 방법이다[11,12,13,14]. 하지만, 그래픽 스트리밍 기술의 자세한 구현 내용은 거의 알려지지 않았다.

3. 원격 렌더링 시스템의 구조

3.1 개괄적 구조

원격 렌더링 시스템은 크게 API 함수 가로채기 모듈, API 함수 전송 모듈, 원격 렌더링 모듈로 구성된다. API 함수 가로채기 모듈과 API 함수 모듈 중 송신 기능은 대체 라이브러리 형태로 구현된다. 즉, 시스템 라이브러리의 기능을 수행하는 대체 라이브러리가 API 함수 가로채기 기능과 송신 기능을 담당한다. 대체 라이브러리는 어플리케이션이 구동되면서 로딩되어 그 기능을 수행한다. 원격 서버에서 구동하는 원격 렌더링 모듈은 렌더링 기능 이외에 어플리케이션으로부터 API 함수에 대한 정보를 수신하는 기능을 담당한다.

3.2 원격 호출 방식

원격 렌더링 시스템을 구성하는 함수 가로채기 모듈은 각 함수의 호출 방식과 그 함수에 대한 원격 렌더링 모듈에서의 호출 방식에 따라 그 구조는 크게 달라진다. 원 함수의 호출 방식이 참조 호출인 경우, 원격 호출에서 참조하는 데이터가 어플리케이션 실행 서버에 존재하는 데이터이기 때문이다. 구체적으로, 값 호출인 경우는 함수 호출에 사용되는 인자를 함수 호출 정보와 함께 원격 렌더링 모듈에 전송하고 호출한다. 참조 호출인 경우는 참조되는 인자의 성격에 따라 원격 호출의 방식을 크게 데이터 복사, 데이터 추출, 포인터 연산, 복합 방식 등 크게 4가지로 분류할 수 있다. ① 데이터 복사는 원 함수의 호출인자가 구조체의 포인터이고 구조체의 데이터가 함수 실행에서 사용하는 경우이다. 원격 호출은 원 구조체의 데이터를 원격 렌더링 모듈에 전송하여 일시적인 구조체를 생성함으로써 가능하다. 일시적으로 생성된 구조체의 포인터를 원격 호출의 인자로 사용한다. IDirect3DDevice9의 멤버함수 HRESULT MultiplyTransform (D3DTRANSFORMSTATETYPE State, CONST D3DMATRIX *pMatrix)가 하나의 예이다. 현재의 월드 행렬 또는 뷰 행렬, 투상 행렬에 새로운 행렬 값을 곱하여 변경하는 함수로써, 두 번째 인자는 행렬을 참조하는 포인터 값이다. ② 데이터 추출 방식은 원 함수의 호출 인자로서 구조체의 포인터가 사용되지만 함수의 목적이 구조체가 나타내는 데이터를 구하고자 함이다. 따라서, 원격 호출 방식은 원격 렌더링 모듈에 일시적인 구조체를 생성하고 이 구조체의 포인터를 함수 호출의 인자로 넘겨준다. 호출 결과 구해진 구조체의 데이터는 어플리케이션 모듈에 전송되는 방식이다. IDirect3D의 멤버함수 HRESULT GetAdapterDisplayMode (UINT Adapter, D3DDISPLAYMODE *pMode)가

2) 마이크로소프트의 함수 가로채기 솔루션

그 예이다. 첫 번째 인자로 지정된 디스플레이 어댑터에 대한 디스플레이모드를 추출하며, 그 결과는 두 번째 인자가 가리키는 구조체에 저장된다. ③ 포인터 연산은 원 함수의 호출에서 호출인자가 포인터로 사용되지만 Direct3D 객체의 포인터를 설정하거나 포인터 값을 구하는 경우에 적용된다. Direct3D 객체는 원격 서버에서 생성된 객체이므로 객체를 가르키는 포인터는 어플리케이션 서버에서 유효하지 않다. 원격 호출을 구현하기 위해서는 객체의 포인터 값을 원격 렌더링 모듈에 전송하여 활용하거나 원격 렌더링 모듈에 일시적인 포인터 변수를 선언하고 이를 활용하여 객체의 포인터 값을 구하고 이를 어플리케이션에 전송하는 것으로 구현한다. IDirect3DDevice9의 HRESULT SetIndices (IDirect3DIndexBuffer9 *pIndexData)는 객체의 포인터 값을 지정하는 함수의 예이며, 같은 인터페이스의 HRESULT GetDirect3D(IDirect3D9 **ppD3D9)은 객체의 포인터 값을 구하는 함수의 예이다. ④ 복합 방식은 위 3가지 방식 중 2가지 이상의 방식이 결합되어 구현되는 원격 호출 방식이다. 예를 들면 원격 렌더링 모듈에 데이터를 복사하고 동시에 객체의 포인터 값을 구하는 경우이다. 대표적인 예가 IDirect3D 인터페이스의 멤버함수 CreateDevice()[3]이다. [Table 4]는 일부 전역함수와 일부 인터페이스의 멤버함수에 대하여 호출 방식과 이에 대한 원격 호출 방식을 구분하여 나타내고 있다. DirectX의 코어 라이브러리는 하나의 전역함수 IDirect3D9 *CreateDirect3D9 (UINT sdkVersion)를 보유하고 있으며, 값 호출 방식이며 원격 렌더링 모듈에서도 값 호출 방식으로 구현된다. IDirect3D 인터페이스는 총 17개의 멤버함수를 보유하고 있으며, 원격 호출 방식은 값 호출, 데이터 복사, 데이터 추출, 포인터 연산, 복합 방식이 각각 9개, 1개, 5개, 1개, 1개이다. 위에서 언급한 4가지 방식이외의 방법으로 원격함수 호출이 이루어지는 경우가 있다. 예를 들면, 확장 라이브러리의 전역함수 D3DXCreateTextureFromFile()[4]는 어플리케이션 서버에 상주하는 파일(매개변수 pSrcFile로 참조하는)로부터 텍스쳐 객체를 생성하는 기능을 담당하며 원격 호출로 그 기능을 수행하고자 한다면, 특별한 처리가 필요하다. 함수가 필요로 하는 파일은 함수가 호출되는 렌더링 서버에 존재하지 않고 어플리케이션 서버에 존재하기 때문이다. 텍스쳐 객체의 생성에 소요되는 파일을 전송하는 대신에 메모리에 읽어 들이고 그 메모리의 내용을 전송하고 이를 대상으로 유사함수 D3DXCreateTextureFromFileInMemory()[5]를 호출한다. 파일 읽기, 파일 전송, 파일 저장, 파일 읽기 등에 많은 시간이 소요되기 때문이다.

[Table 4] Remote Call Methods of Called Functions

Librarry	Call by value	Call by reference			
		Data copy	Data extract	Pointerop eration	Comb ined
Core Lib. Func.	1	0	0	0	0
IDirect3D	9	1	5	1	1
IDirect3D Device9	26	23	26	36	8
Ext. Lib. Mesh Func.	2	5	0	15	18
ID3DX Mesh	14	7	0	8	0
ID3DX Effect	34	13	22	10	0

3.3 대체 라이브러리의 구성

본 원격 렌더링 시스템은 API 함수 가로채기 방법으로 대체 라이브러리 기술을 사용한다. 시스템 라이브러리가 제공하는 함수를 별도로 구현하고

3) HRESULT CreateDevice(UINT Adapter, D3DDEVTYPE DeviceType, HWND hFocusWindow, DWORD BehaviorFlags, 3DPRESENT_PARAMETERS *pPresentationParameters, IDirect3DDevice9** ppReturnedDeviceInterface)

4) HRESULT D3DXCreateTextureFromFile (LPDIRECT3DDEVICE9 pDevice, LPCTSTR pSrcFile, LPDIRECT3DTEXTURE9 *ppTexture);

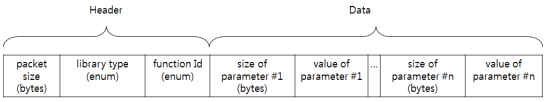
5) HRESULT D3DXCreateTextureFromFileInMemory (LPDIRECT3DDEVICE9 pDevice, LPCVOID pSrcData, UINT SrcData, LPDIRECT3DTEXTURE9 *ppTexture);

이들을 통합하여 라이브러리 형태로 구성한다. 새로운 라이브러리로 시스템 라이브러리를 대체하면 어플리케이션이 구동될 때, 시스템 라이브러리 대신 로딩되어 그 기능을 수행한다. 원 시스템 라이브러리는 복사본으로 유지되고, 대체 라이브러리가 원 함수의 기능을 필요로 하는 경우 복사본의 원 함수를 호출할 수 있도록 한다. 대체 라이브러리는 API 함수 가로채기 기능 이외에 API 함수 전송 모듈의 송신 기능을 담당한다.

어플리케이션이 호출하는 API 함수에 대한 추출된 정보는 함수 별 패킷을 구성하여 원격 렌더링 모듈에 즉각 송신된다. 각 패킷은 헤더 부분과 데이터 부분으로 구성한다. 헤더는 전체 패킷 크기, 라이브러리 종류 식별자, 함수 식별자로 이루어진다. 데이터 부분은 호출되는 함수의 매개변수 데이터로 이루어지며, 매개 변수 자료 크기, 매개 변수 값으로 이루어진다([Fig. 1] 참조). 크기와 식별자에 대한 정보는 운영체제에서 지원하는 정수 자료형을 활용하여 지정한다.

3.4 원격 렌더링 모듈

수신된 API 함수의 호출 정보를 이용하여 해당 함수를 호출하고 이를 통하여 프레임 이미지를 생성한다. 각 함수의 호출 결과는 어플리케이션 서버에 피드백하여 어플리케이션의 실행에 참조하도록 한다.

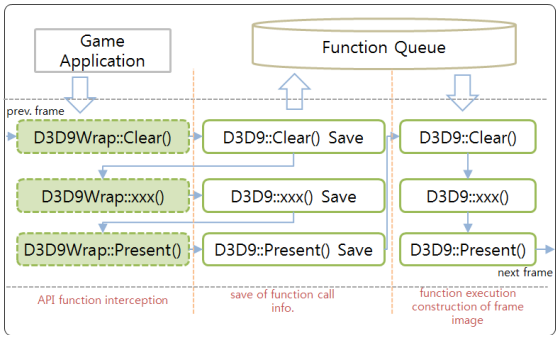


[Fig. 1] Construction of Transmission Packets for Information of Called Functions

GPU가 대규모 데이터의 빠른 연산을 위해 파이프라인 구조를 갖는 특성을 고려하여 하나의 프레임 이미지 생성에 필요한 함수는 함수 큐에 저장하고 이를 연속적으로 실행한다. 하나의 프레임 이미지 생성이 완료되면 다음 프레임의 함수호출 정보를 함수 큐에 저장하여 이전 프레임의 함수호출 정보를 대체한다([Fig. 2] 참조).

4. 실 험

본 논문에서 제안하고 있는 원격 렌더링 시스템의 기술적 가능성을 검증하기 위하여 샘플 어플리케이션에 적용하고 성능의 측정 및 분석을 수행한다. 각 프레임의 이미지를 생성하기 위한 API 함수의 전송 시간과 각 프레임의 이미지를 생성하기 위해 소요되는 시간 즉, 렌더링 시간을 측정하고 그 결과를 분석한다.



[Fig. 2] Control Flow of Remote Rendering System

4.1 샘플 어플리케이션

본 실험에서는 DirectX 9.0c SDK(2006년 4월 배포)에서 제공하는 'ShadowMap' 샘플 어플리케이션을 변형하여 본 연구에서 구현한 원격 렌더링 시스템에 적용한다. 샘플 어플리케이션은 쉐도우 맵을 활용하여 그림자를 표현하기 위하여 프로그래머블 쉐이더(Programmable Shader) 프로그램을 별도로 작성하여 실행한다. 구현의 용이함을 위하여 원래의 온스크린디스플레이(OSD, On-screen Display)와 디스플레이 장치에 대한 설정 UI를 제외하였다.

[Table 5]은 샘플 어플리케이션이 호출하는 함수와 각 인터페이스의 멤버함수이다. 코어 라이브

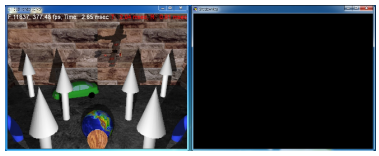
러리에서는 유일한 전역함수 CreateDirect3D9()[6]가 호출된다. 또한, 코어 라이브러리에 속하는 IDrect3D 등 6개의 인터페이스에 속하는 멤버함수 중 58개의 함수가 호출된다. 이는 코어 라이브러리가 제공하는 총 17개의 인터페이스 중 6개의 인터페이스가 활용되는 것이다. 6개의 인터페이스가 제공하는 203개의 멤버함수 중 58개의 함수를 사용한다. 확장 라이브러리에서는 D3DXLoadMeshFromX()[7] 함수 등 총 6개의 전역함수가 호출된다. 또한 확장 라이브러리에 속하는 ID3DXMesh 등 3개의 인터페이스에 속하는 멤버함수 중 35개의 함수가 호출된다. 이는 확장 라이브러리가 제공하는 총 27개의 인터페이스 중 3개의 인터페이스가 활용되는 것이다. 3개의 인터페이스가 제공하는 113개의 멤버함수 중 29개의 함수를 호출한다.

[Table 5] Called Global Functions and Interface Member Functions of The 'ShadowMap' Program

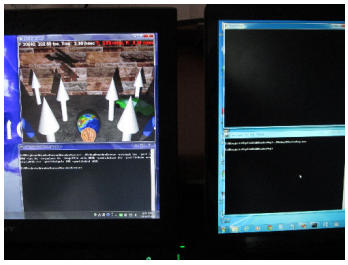
Interfaces		In use	Not used
Core Library	core functions	1	0
	IDirect3D	13	4
	IDirect3DDevice9	31	88
	IDirect3DVertexBuffer9	3	11
	IDirect3DIndexBuffer9	3	11
	IDirect3DTexture9	3	19
	IDirect3DSurface9	5	12
	sum	59	145
Ext Library	Ext. Mesh fucntions	1	39
	Ext. Effect functions	1	10
	Ext. Texture functions	2	46
	Ext. misc. functions	2	5
	ID3DXMesh	11	18
	ID3DXEffect	16	63
	ID3DXBuffer	2	3
	sum	35	184

실험은 샘플 어플리케이션과 대체 라이브러리가 구동하는 어플리케이션 서버와 원격 렌더링 모듈이 구동하는 렌더링 서버가 물리적으로 분리된 환경에서 수행된다. 어플리케이션 서버는 Intel Core i7 3630QM(2.4Ghz) CPU를 탑재하고 있다. 운영제제로 MS Windows 7 Professional K(64bit)를 운용하고 있으며 4.0GB의 주 메모리를 탑재하여 사용한다. 그래픽 어댑터로는 NVIDIA GeForce GT 640M를 사용한다. 렌더링 서버는 Intel Core i7 3520M(2.9Ghz) CPU를 탑재하고 있다. 운영제제로 MS Windows 7 Professional K(64bit)를 운용하고 있으며 8.0GB의 주 메모리를 탑재하여 사용한다. 그래픽 어댑터로는 NVIDIA GeForce GT 640M LE를 탑재하여 사용한다.

[Fig. 3]은 샘플어플리케이션과 렌더링 모듈을 단일 서버에 설치하고 실행한 화면의 예이다. 온스크린디스플레이를 별도로 구현하여 렌더링 모듈에서 FPS, 프레임 별 패킷 전송 시간, 프레임 별 렌더링 시간 등의 성능 지수를 화면 상단에 표시한다. [Fig. 4]는 어플리케이션 서버와 렌더링 서버를 물리적으로 분리하여 실행한 화면의 예이다.



[Fig. 3] Screen Capture 1(Local System)

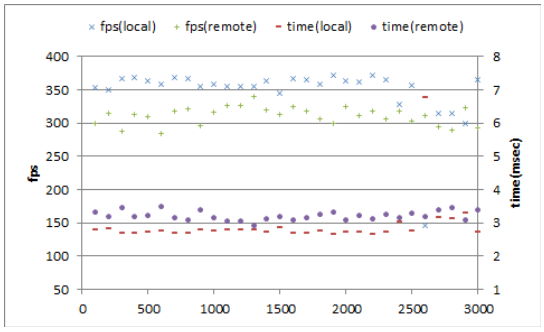


[Fig. 4] Screen Capture 2(Distributed System)

6) IDirect3D9 *CreateDirect3D9 (UINT sdkVersion)
7) HRESULT D3DXLoadMeshFromX (LPCTSTR, DWORD, LPDIRECT3DDEVICE9, LPD3DXBUFFER*, LPD3DXBUFFER*, DWORD*, LPD3DXMESH*)

4.2 시스템 성능

원격 실행을 적용하는 경우, 하나의 시스템에서 단독으로 실행하는 경우보다 성능이 약간 떨어진다. 평균적으로 단독 실행인 경우, 프레임 당 평균적으로 2.96msec가 소요된다. 원격 실행인 경우, 프레임 당 3.22msec가 소요된다. 8.9%의 추가적인 시간이 소요된다. 이를 초당 프레임 수로 환산하면, 단독 실행인 경우 348.04fps이고, 원격 실행인 경우 310.98fps이다. 이는 10.6%의 성능 감소를 의미한다([Fig. 5] 참조). 원격 렌더링을 구현하더라도 하나의 프레임 이미지를 생성하는데 소요되는 시간이 실시간 게이밍 서비스의 지연시간 50msec의 약 6.4%의 수준에 불과하다.



[Fig. 5] Performance Comparison of Standalone Execution and Remote Execution

4.3 API 함수 전송 시간

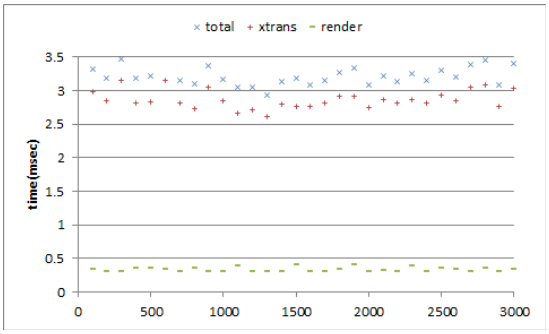
API 함수 전송 시간은 평균적으로 2.87msec가 소요되고 있다. 이는 프레임 당 소요시간이 평균적으로 3.22msec가 소요되고 있기 때문에 전체 프레임 당 소요시간의 89.0%를 차지한다([Fig. 6] 참조). 네트워크 패킷의 전송에 상대적으로 많은 시간을 소요하지만, 전체적인 소요시간이 게임의 실시간 반응성을 만족시키기 때문에 대부분의 게임 어플리케이션에 적용 가능하다.

5. 결 론

실험의 결과에서 확인된 바와 같이 본 논문이 제안하고 있는 실시간 원격 렌더링 시스템은 프레임 별 렌더링 시간과 API 함수 전송 시간을 고려할 때, 3D 어플리케이션의 원격 서비스가 기술적으로 가능한 것으로 확인된다.

본 연구의 결과는 다음과 같은 의의가 있다. ① 지금까지 거의 알려지지 않았던 원격 렌더링 시스템의 구현에 있어서 필요한 세부사항을 확인할 수 있다. ② 물리적으로 분리된 렌더링 서버의 GPU를 활용하기 때문에 일반적인 GPU 가상화보다 많은 응용 분야를 갖는다. GPU 가상화는 단일 시스템에 존재하는 다수의 VM이 GPU를 공유하는 반면에, 본 연구의 결과물은 원격 서버의 GPU를 활용하기 때문에 보다 폭 넓게 활용할 수 있다.

본 연구는 샘플 어플리케이션의 구현에 사용되는 API 함수에 대해서 원격 호출을 구현하여 원격 렌더링 시스템을 구성하였다. 일반적인 Direct3D 어플리케이션을 지원하기 위해서는 Direct3D의 모든 인터페이스의 멤버함수와 모든 전역함수에 대한 원격 호출이 구현되어야 한다. 여러 GPU 가상화 방법 중 H/W 가상화 방법은 GPU의 완전한 기능과 성능을 공유할 수 있도록 한다. 향후 연구로 H/W 가상화 방법을 원격 W/S의 GPU에도 구현할 수 있다면 원격 W/S의 온전한 기능과 성능을 다수의 단말기가 공유할 수 있다.



[Fig. 6] Per Frame Analysis of Required Time

REFERENCES

[1] NAVER Developer Blog (07-22-2012), "Understanding of 3G Network", Retrieved from http://helloworld.naver.com/helloworld/111111.

[2] Uplusblog (07-22-2014), "Broadband LTE-A, U LTE8 x3 Speed", Retrieved from http://blog.uplus.co.kr/1847.

[3] J.H.Park, "5G Servvice to Provide New Values to Costumers", TTA Journal, Vol.152, pp.46~51, 2014.

[4] C.G.Lim, S.S.Kim, K.I.Kim, J.H.Won, C.J.Park, "Technology Trends of Cloud Computing-based Game Streaming", Electronics and Telecommunications Trends, Vol.26 No.1, pp.47-56, 2011.

[5] Choong-Gyoo Lim, "A 2-Tier Architecture for Real-time Multiple Rendering", Korea Game Society, Vol.10 No.2, pp.13-22, 2012.

[6] H. Lagar-Cavilla, N. Tolia, M. Satyanarayanan and E. de Lara, "VMM-independent graphics acceleration", VEE 07(3rd International Conference on Virtual Execution Environments), pp.33-43, 2007.

[7] M. Dowty, J. Sugerman, "GPU virtualization on VMware's hosted I/O architecture", ACM SIGOPS Operating Systems Review, Vol. 43, No. 3, pp.73~82, July 2009.

[8] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson and J. Leigh, "High performance dynamic graphics streaming for scalable adaptive graphics environment," Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, FL, November, 2006, pp.108.

[9] OnLive, www.onlive.com

[10] Gaikai, www,gaikai.com

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

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

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

[14] 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.

[15] D.H.Lee (07-18-2012), "LG Uplus Launches the 'C-games' Cloud Game Marget", Digital Daily, http://www.ddaily.co.kr/news/article.html? no=9317

[16] W.C.Joung (07-23-2013), "KT 'Wiz Game', Start of Cloud Game Service", This Is Game.com, Retrieved from http://www.thisisgame.com/webzine/news/nboard/4/?n=45409.

[17] Y.M.Baek (05-20-2013), "SK Broadband Launches the 'Btv' Cloud Streaming Service", NEWSis, Retrieved from http://www.newsis.com/ar_detail/view.html?ar_id=NISX20130520_0012096034&cID=10402&pID=10400.

[18] W, Wade, I. Williams, "NVIDIA GPU Virtualization", GTC 2012, 2012.

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

[20] I. Ivanov (12-02-2002), "API hooking revealed", Retrived from http://www.codeproject.com/Articles/2082/API-hooking-revealed.

[21] www.x.org

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

임 충 규 (Lim, Choong-Gyoo)

1999년 2월-2011년 2월 한국전자통신연구원(ETRI) 책임연구원
2011년 3월-현재 성공회대학교 컴퓨터공학과 교수

관심분야 : 기하모델링, 컴퓨터 그래픽스, 컴퓨터 게임