Sponsored Feature : 진화하는 게임 – 실험적 클라우드 기반의 레이 트레이싱

(Sponsored Feature: Changing the Game – Experimental Cloud-Based Ray Tracing)

대니얼 폴(Daniel Pohl)

가마수트라 등록일(2011. 03. 30)

http://www.gamasutra.com/view/feature/6322/sponsored_feature_changing_the_.php

게임에서의 실시간 레이 트레이싱은 최근에 정점을 찍고 있지만, 유망한 수준의 쉐도우, 반사 및 라이팅 효과는 표준의 렌더링에서는 가능하지 않다. 가장 큰 장애물은 복잡한 계산을 필요로 하고 있다는 것 때문인데, 게임에서의 사용이 극히 제한적이었다는 배경도 가지고 있다. 그러나 클라우드는 이것을 바꾸어 놓을지도 모른다.

클라우드 기반의 렌더링은 게임 월드에서 모멘텀을 얻고 있다. (GDC에서, Gaikai 와 OnLive는 서버 베이스의 게임을 선보였다.) 그래서 서버 사이드 레이 트레이싱이 게임에 활용가능하다고 할 수 있지 않겠는가? 인텔의 스폰서를 받고 있는 이 글에서, 대니얼 폴은 레이 트레이싱 게임 기술의 선도 전문인으로써, 클라우드 기반의 실시간 레이 트레이싱의 장점과 Intel® MIC Architecture를 활용하여 클라우드 기반의 렌더링 접근법을 시연하고자 하고 있다.

더 많은 대니얼의 작업을 보고자 한다면 Quake Wars Gets Ray Traced을 읽고 http://www.wolfrt.de/. 웹사이트를 방문하라.

-- Orion Granatir

서문

컴퓨팅 과정에서 게임을 변화시키는 요인은 클라우드에 기반한다. 단어가 때로는 과용하게

사용되기도 하지만, 컨셉 자체는 기업이나 소비자 모두에게 흥미로운 이득을 제공한다. 클라우드를 활용하는 것에 대한 예는 컴퓨터 게임과 관련 있다.

OnLive와 Gaikai 같은 기업은 게임 그 자체를 클라우드 서버에서 서비스할 수 있도록 하는 비즈니스를 선보이고 있다. 게임과 클라이언트 사이의 상호작용을 처리하면서, 서버는 압축된 렌더링 이미지를 이용자에게 다시 보낸다. 클라우드 기반의 렌더링 접근은 장점이 아주 많다. 가령, 이용자는 게임을 설치하기 위해 기다릴 필요가 없고, 최신 상태의 게임 패치를 받을 걱정을 할 필요도 없다. 이용자의 컴퓨터의 하드 디스크 용량이 아주 적어도 상관없다. 게임은 클라이언트 편에서 복사될 수 없기 때문에, 게임 DVD를 컴퓨터에 넣는 것처럼 불법 복제물 체크를 할 필요도 없다. 개발자에게는 또한 게임의 데모 버전을 쉽게 보여줄 수 있는 창구가 되며, 제한된 시간만 플레이를 할 수 있도록 하여 소비자가 전체 게임에 대해 첫 인상을 가질 수 있는 기회를 준다.

또 다른 장점은 게임 클라이언트가 많은 다른 OS에서 구동되고 이로 인해 각기 다른 플랫폼에 맞게 포팅하는데 소요되는 시간을 줄일 수 있게 된다. 복잡한 계산이 서버에서 이루어지기 때문에 넷북, 태블릿 및 스마트폰과 같은 다양한 플랫폼에서 최신의 게임이 구동될 수 있다. 어떠한 장치에서라도 전문적으로 그래픽이 복잡한 최고의 게임을 구동할 수 있는 가능성을 열어주게 된다. 최근의 게임은 대체로 게이머가 기대하고 있는 수준의 퍼포먼스를 갖추기 위해서 일반적인 사양의 그래픽 카드에 맞도록 하드웨어 렌더링을 이용하고 있다. 이 접근법 역시 장점이 많지만, 게임 개발자가 그들의 게임에서 활용할 수 있는 선택을 제한하게 된다. 포인트 렌더링, 픽셀 렌더링 또는 레이 트레이싱과 같은 다른 렌더링 알고리즘이 있을 수도 있고, 이것은 게임이 좀더 사실적으로 보이도록 해 줄 수 있다.

레이 트레이싱은 반사, 굴절, 그림자와 같은 정확한 계산된 효과로 실사와 같은 그래픽을 만들어 내는 물리 법칙을 이용한 기술을 말한다. 클라우드 게임 모델은 실시간으로 레이 트레이싱을 가능하게 하는데 필요한 최신의 하드웨어에 좀더 많은 사람들이 접근할 수 있도록 하는 방법을 제공한다.

최근에, Intel Labs는 멀티 코어 Intel® processor에서 구동되는 실시간 레이 트레이싱 엔진

개발에 점진적인 발전을 보이고 있다. 2008년, 우리는 쿼드 코어 CPU를 이용한 서버를 기반으로 둔 Intel® Xeon® processor에서 초당 15-20 프레임을 구동할 수 있는 Quake Wars: Ray Traced를 볼 수 있었다. 최적화를 시행한 몇 달 후에, 우리는 똑 같은 네개의 소켓 서버를 이용한 데모를 볼 수 있었다. 그러나 새로 업데이트 된 식스 코어 Intel® Xeon® X7460 processors는 20-35 fps 였다. 그 다음해 Intel processor는 똑 같은 프레임을 달성할 수 있는 투 소켓을 가진 워크 스테이션 시스템으로 넘어갔다. 지속적으로 퍼포먼스가 증가되는 상위 코어 코넛과 Sandy Bridge 라는 코드 네임의 Intel® microarchitecture와 같은 새로운 아키텍쳐를 보게 될 것임이 분명하다.

이 알고리즘을 위한 플랫폼이 2010년 5월에 발표된 Intel® Many Integrated Core Architecture (Intel® MIC Architecture)으로 다가오고 있다. 레이 트레이싱이 매우 병행적인 알고리즘이기 때문에, Intel MIC Architecture는 퍼포먼스 측면에서 큰 성과를 가져오게 할 것이다. 고성능 컴퓨팅 시장의 병행적 어플리케이션 및 데이터 센터에 가용한 수 많은 코어를 증가시켜서 가능하게 할 것이다. 이러한 사실들이 이 글의 주제를 이끌어 낸다 : 클라우드 기반의 게이밍 모델에 따라 Intel MIC Architecture를 함께 개발하여 실시간 레이 트레이싱으로 좀더 발전되고 사실적인 이미지 렌더링을 가능하게 한다는 것이다.

Ray Tracing 개괄

레이 트레이싱은 아주 사소한 차이로, 자연에서 찾을 수 있는 빛의 광선을 자극시키는 렌더링 알고리즘이다. 본질적으로, 광선은 빛의 원천(가령, 태양)에서 기인하여 종국에는 인간의 눈을 비춘다. 레이 트레이싱 알고리즘은 역으로 이 순서를 따른다 : (가상 카메라 처럼) 눈에서 장면으로. 이러한 광선이 지형물의 물체를 비추는 지점이 계산된다. 그리고 나서, 쉐이더 프로그램이 표면을 구성하기 시작한다.

가상 카메라에서 나온 광선이 자동차를 비추고 집을 향해 반사된다. 쉐도우 레이는 빛의 원천을 향해 자동차에서 나아간다.

이 과정을 통해, 반사와 투과와 같은 물질적인 특징이 평가된다. 텍스쳐의 색깔이 덧입혀질 수 있고 비추임, 반사 및 굴절을 테스트 하는 두번째 광선이 여기서 추적될 수 있다.

레이 트레이싱은 이미 전문적인 그래픽 시장에서는 흔하게 사용되어지고 있다. 이것은 보통 오프라인(즉, 상호작용적이지 않은) 업무에서 행해지는데, 엄청난 시간을 하나의 이미지에 소비하는 그런 업무에 사용된다. 인터랙티비티의 정도를 가질 수 있도록 하기 위해서, 여러 번 정제되는 최종 이미지의 프리뷰가 존재한다. 때로는, 큰 클러스터가 인터랙티비티를 가지기 위한 해결책이 되었었다.

자동차 산업은 레이 트레이싱을 전문적으로 활용한 중요한 예이다. 자동차가 설계되기 전에, 실사와 같은 방식으로 첫 모습을 미리 볼 수 있다는 것은 매우 중요하다. 이 과정을 통해, 제조사는 원하는 특징을 가진 자동차 모델을 만들 수 있고, 만약 자동차의 특정한 물체나 그림자가 운전자를 방해할 지도 모르는 반사를 야기시킨다면, 디자인 단계에서 쉽게 결함을 발견할 수 있다. 헤드라이트와 같은 물체를 모델링하기 위해서, 물체 안에서 튀어 나오는 다양한 광선이 있는 복잡한 조명 계산이 해결되어야만 한다. 레이 트레이싱은 물리적으로 정확한 이미지를 얻을 수 있는 가장 좋은 해결책임이 증명 되었다.

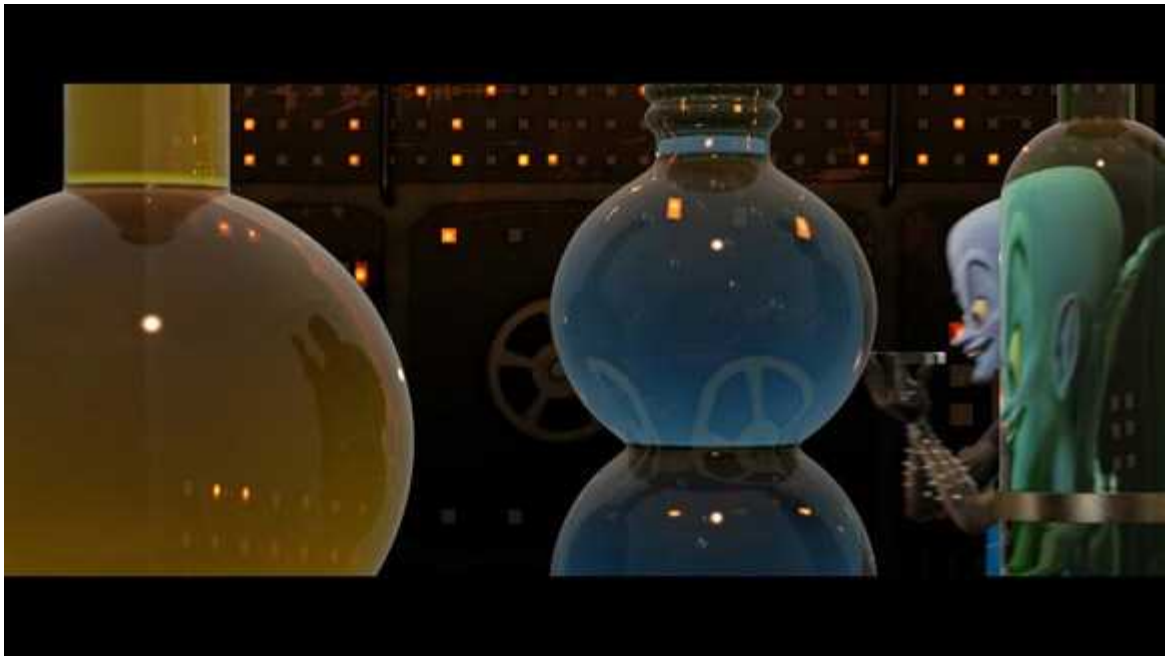
오프라인 렌더링 된 자동차 헤드라이트 모델

레이 트레이싱의 또 다른 어플리케이션은 의료 시각화 분야에 있다. CT에 활용되는 스캐닝 장비나 MRT는 인간의 뇌나 두개골 같은 대상의 3D 데이터를 창조한다. 이 체적 측정 정보는 컴퓨터 스크린에 디스플레이 되어야만 한다. 이것을 위한 많은 다른 방법이 있지만, 레이 트레이싱이 가장 정확하다.

광선을 이용해 시각화 된 체적 측정 데이터 두개골

(Source: http://en.wikipedia.org/wiki/File:High_Definition_Volume_Rendering.JPG)

영화 산업은 레이 트레이싱은 강건하고 안정적인 방식으로 굴절과 반사를 해결하고자 알고리즘을 가장 일반적으로 활용하고 있는 분야이다.



레이 트레이싱을 이용한 특수 효과를 보여주고 있는 MEGAMIND™ (©DreamWorks Animation)의 영화 이미지

Intel® MIC Architecture Ray Tracing

아래의 예는 클라우드 기반의 게이밍 접근과 Intel MIC Architecture를 이용한 레이 트레이싱 렌더링을 복합적을 활용한 것을 평가한 것이다.

Hardware Setup

이 프로젝트를 위해 네 개의 "서버" 머신이 클라우드를 재현하기 위해 사용되었다. 각각의 구성은 다음과 같다 :

Motherboard: Intel® DX58SO (code name Smackover)

CPU: Intel® Core™ i7-980X processor (6 cores, 2 threads per core, 3.33 GHz)

Intel code name Knights Ferry PCIe card (32 cores, 4 threads per core)

Gigabit Ethernet

Intel 코드 네임 Knights Ferry는 Intel MIC Architecture의 1세대 개발 플랫폼이다. 이것은 1.2 GHz 속도에 32 코어 칩을 가진 PCIe 카드를 포함하고 있다. 개발 플랫폼은 일반적인 툴과 프로그래밍 언어로 프로그래밍 된다. Intel MIC Architecture 기반의 생산품을 위해 조금 더 발전한 계획이 있는데, 코드 네임 Knights Corner이며, 이것은 22 nm 제조 기술을 이용할 예정이며, 칩에 50 코어 이상을 가질 수 있게 될 것이다.

얇은 클라이언트(게이머의 장비를 재현하는)로써, 작은 노트북이 선택되었다. 사양은 다음과 같다 :

CPU: Intel® Core™2 Duo processor P9600 (2 cores, 2.66 GHz)

13-inch screen at 1280x800

Gigabit Ethernet

Software Setup

게임 콘텐츠는 id Software와 Raven Software에 의해 만들어진 *Wolfenstein* (2009) 독일 버전이 선택되었다.

레이 트레이싱은 Intel Labs에 의해 개발된 실험 엔진으로 수행되었다. 기하학적 구조, 카메라 데이터, 텍스쳐, 변화된 상태 등등의 과정을 거치기 위해서 규정된 API에서 프로그램 가능하도록 되었다. 게다가, HLSL 같은 쉐이딩 언어가 편리한 방식으로 고 품질의 쉐이딩 코드를 작성할 수 있도록 지원되었다.

레이 트레이싱 엔진은 두 부분으로 되어 있다. 한 쪽은 CPU에서 완전히 구동되고 콘텐츠를 업로드 하고, 상태를 관리하며, 렌더링 명령을 보내기 위해서 Knights Ferry 카드와 커뮤니케이션 하는데 사용된다. 다른 부분은 Intel MIC Architecture에서 실행된다. 16-wide SIMD을 활용하는 특수 코드와 일반적인 Intel® architecture CPU와 유사한 방식으로 실행된다. 렌더러는 32코어 중 31을 이용하는데, 마지막 것은 남겨두어야 한다. 드라이버가 필요로 하는 어떠한 업무라도 수행할 수 있도록 하기 위해서이다. 각각의 코어가 4개의 쓰레드를 가지게 되면, 124개의 쓰레드가 가능하다는 것을 의미한다. 이것들은 역동적으로 Knights Ferry에 다른

업무를 할당시키고, 가장 큰 부분이 컬러 픽셀과 레이 트레이싱을 통한 실질적인 렌더링 업무를 담당하게 된다. 두 번째로 큰 부분이 내부 가속 구조를 업데이트 하는데 이용된다. 이로 인해 게임의 기하학적 구조의 재현을 유지하게 된다. (플레이어의 동작, 아주 작은 업데이트 등) 장면의 역동적인 변화는 새로운 정보를 가지고 업데이트가 잘 되도록 하는 가속 구조를 필요로 한다.

클라이언트-서버 렌더링

The Intel® Core™ i7 processor는 (예를 들어 플레이어가 현재 선택하고 있는 무기와 같은 플레이어의 포지션과 같은) 모든 단계를 업데이트하고 진행하도록 하는 게임 엔진을 구동시킨다. 그리고 Knights Ferry의 렌더링과 CPU 상의 인터페이스를 이미 언급한 바와 같이 구동할 수 있도록 레이 트레이싱과 커뮤니케이션 할 수 있도록 한다. 또한 TCP/IP를 통해 클라이언트에게 렌더링된 픽셀을 이동시킨다.

클라이언트는 이용자 인터페이스(키보드, 마우스 움직임)을 처리하고 (Up 버튼을 눌러서 플레이어의 위로 움직이게 만드는 것과 같은) 행동을 평가한다. 업데이트된 게임 상태는 서버로 보내진다. 클라이언트는 서버로부터 픽셀 데이터를 받고 이것을 디스플레이 한다.

멀티플 하드웨어 유닛의 파워 계산을 결합하기 위해서, 4개의 서버와 같은 경우에는 사용될 수 있는 다른 방법이 있다. 좀더 세부적으로 2가지를 언급한다 :

> **모든 서버를 아우르는 타일 기반의 렌더링 디스트리뷰팅**. 이 방법은 레이 트레이스 된 이미지를 작은 타일 (32x32 또는 64x64 pixels)로 렌더링의 작업을 쪼개서 특정한 서버에 할당시킨다. 이 접근법의 장점은 시간 지체가 적다는 것이다. 장비는 가능한한 빠르게 하나의 프레임을 마무리 짓기 위해서 함께 작업할 것이다. 문제점은 서버 사이의 정확한 로드 밸런싱을 위해 똑똑한 알고리즘이 필요하다는 것이다. 때로 타일이 다른 것(가령, 어떤 지형적인 특성 없이 하늘을 디스플레이 하는 것이 매우 빠르다) 보다도 훨씬 더 빨리 계산될지도 모른다. 그러므로 장비의 한 부분이 이미 작업을 끝내고도 다른 부분이 작업을 마칠 때 까지 기다리고 있어야 하는 일이 발생할 수 있다.

이를 해결하기 위해서, 업무 훔치기와 같은 접근법이 있다. 이것은 한가한 쓰레드가 다른 바쁜 쓰레드의 파이프라인으로부터 작업을 가져올 수 있도록 하는 것으로 수행이 잘 이루어져야만 한다.

대안적 프레임 렌더링. 이 접근법을 이용하면, 구체적인 프레임이 구체적인 장치에 할당되도록 할 수 있다. 이것의 장점은 각각의 서버에 할당하는 것이 용이하다는 것이다. 대부분의 경우에 연속적인 프레임을 계산하는데 걸리는 시간은 크게 다르지 않다 ; 그러므로 작업은 균형이 잘 맞추어져 진다. 문제점은 이 방법은 부가적인 랙을 가져올 수 있다는 것이다. 4개의 서버인 경우에, 클라이언트 장비에 디스플레이되는 것에 3 프레임 정도의 지연을 발생시킬 수 있다. 60 Hz 프레임 속도일 경우, 지연은 50 milliseconds (ms) 정도 발생될 것이다. 우리의 데모 세팅에서 더블 버퍼링을 이용할 경우, 일곱 프레임 (117 ms) 지연으로 이어진다.

두 번째 방법을 통해 부가적인 지연을 가지게 될 수 있기 때문에 상업적인 실행방법으로는 타일 기반의 접근법을 활용하기를 추천한다. 그럼에도 불구하고, 이 글의 목적을 위해서, 우리의 첫 번째 실행법은 향후 수정될 것을 염두해 두고, 타일 기반의 방법을 활용하여 대안적 프레임 렌더링을 활용하도록 한다.

압축

스크린 상에 보이는 이미지를 메모리에서 그래픽 장비로 업로드 하기 위해서, 거의 모든 요즘 디바이스는 붉은색, 녹색, 파란색 및 알파의 RGBA 채널을 가진 이미지를 필요로 한다. 이 색깔은 각각 8 비트의 정보를 가지고 있다. 1280x720의 이미지의 경우, 3600 KB의 메모리 정보를 잡아 먹는다. 알파 채널은 유용한 정보에 기여하지 못하기 때문에, 쉽게 2700 KB로 줄어들 수 있다. 60 Hz의 속도일 경우, 1초당 약 158.2 MB (1264 Mbit) 비율의 데이터가 필요해 진다. 그러므로 Gigabit Ethernet의 대역폭을 훨씬 초과하게 된다. 적당한 데이터를 처리하기 위한 압축의 형태가 필요하다.

압축을 이용하는 것은 서버 측면으로는 인코등을 위해서, 클라이언트 측면에서는 디코딩을

위한 부가적인 계산을 할 수 있도록 해 준다. 서버에는 압축을 하기 위한 수 많은 자원들이 있다. 이러한 환경에서, 6-core Intel® Core™ i7 processor는 멀티-쓰레드 인코딩 알고리즘을 수행한다. 그렇지 않으면, 압축은 Knights Ferry 카드에서 직접적으로 수행되어질 수도 있다. 반면에, 클라이언트는 좀더 가벼운 시스템이 될 것이며, 이미지를 디스플레이 하기 전에 디코드 하는데 좀더 시간을 필요로 하게 될 것이다.

이용될 수 있는 다양한 코덱이 있다. OnLive는 그들 스스로의 코덱을 이용하여 비디오 스트리밍을 할 수 있도록 특수한 인코딩 칩을 이용하고 있다고 밝힌 바 있다. YouTube와 Blu-ray videos에서 찾을 수 있는 Gaikai는 H.264 compression codec을 이용하고 있다. 1280x720크기로 5 Mbit/s의 속도로 비디오 스트리밍을 하는 것은 합리적인 수준의 품질이지만, 또한 향상을 위한 여지를 가지고 있다. 이용할 만한 대역폭이 있다면, 대부분의 코덱은 더 좋은 영상 품질을 위해서 몇몇의 파라미터를 변경시켜서 쉽게 이용할 수 있다.

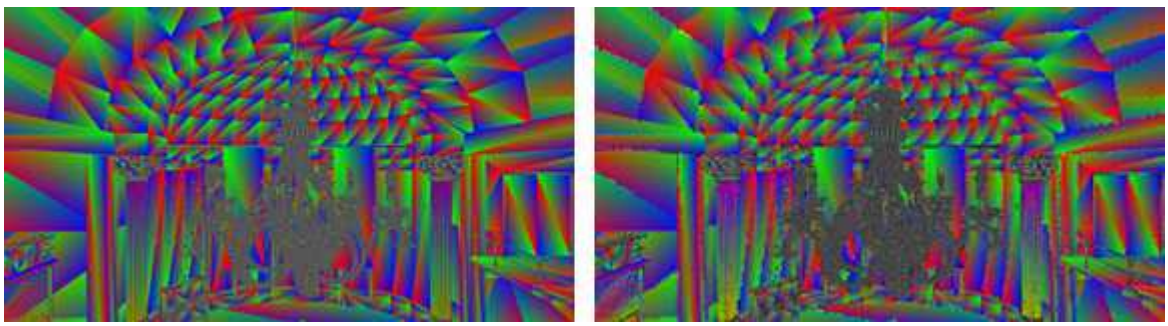
OnLive를 이용한 Dirt 2 스크린 샷

광대역폭의 Gigabit Ethernet 연결이라는 최상의 네트워크 시나리오로 압축에 필요한 요소들을 줄이고 최상의 시각적인 결과물을 달성할 수 있다. 이 대역폭으로 우리는 각각의 이미지에

기반을 둔 다른 형태의 압축을 이용할 수 있는데, 움직임의 독립성이 안정적으로 보일 수 있는 고 품질의 영상을 결과물로 발생시킨다. 이용되는 알고리즘은 DXT1이다. 이 알고리즘은 1:8의 비율로 RGBA 데이터를 압축시키고 게임에서는 .dds 파일 컨테이너를 통해 이용된다. Intel® Streaming SIMD Extensions (Intel® SSE)을 이용한 압축과 압축해제를 빠르게 실생하는 방법이 있다. 그리고 Intel® Integrated Performance Primitives (Intel® IPP) 7.0 라이브러리에서 멀티-쓰레드를 활용할 수 있다. 또한 일부 그래픽 칩은 어떠한 수동적인 압축 해제 없이도 DXT1 압축 내용물을 디스플레이 할 수 있다. DXT1을 적용한 결과물은 일반적인 게임 장면에서의 오리지널 소스에서 알아차리기 힘들 정도의 손실만을 가져온다.

좌 : 압축되지 않은 이미지. 우 : DXT1으로 압축된 이미지

물론, 가공물이 너무 뚜렷해 질 수 있는 일부의 특수한 경우가 있다. 대표적인 예는 빨간색, 녹색 및 파란색의 꼭짓점을 가진 쉐이드 된 트라이앵글을 보여주는 디버그 뷰이다. DXT1 알고리즘은 이러한 자연스럽지 못한 이미지에는 적합하지 않다.



좌 : 압축되지 않은 이미지. 우 : DXT1을 이용하여 압축한 이미지

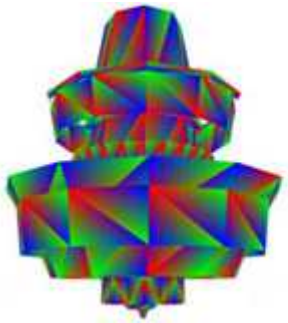
울펜스테인 : 레이 트레이스

게임에서 레이 트레이스를 이용할 경우 가질 수 있는 장점을 보여주기 위해서, 오리지널 게임의 몇 가지 요소가 울펜스테인에 맞게 변경되었다 : 레이 트레이스 데모. 슈퍼샘플링을 통해서 렌더링 된 스크린 샷에 주목해 주기를 바란다. 이것이 적용되지 않은 이미지는 나중에 소개될 것이다.

기하학적 구조

레이 트레이싱 엔진은 전형적으로 가속 구조를 이용하는데, 좀도 효율적으로 계산 될 수 있도록 하기 위해서 오브젝트 간에 빛의 상호작용이 가능하도록 장면을 배열하는 기하학적 구조를 통해 계층을 형성하는 것을 의미한다. 이것으로, 몇 개의 삼각형을 이용하여 대수적으로 빛을 증가시키는데 필요한 계산을 수행하고, 정적인 콘텐츠를 좀더 멋있게 보이도록 할 수 있다. 그러므로, 향상된 기하학 구조적 디테일은 성과물에 작은 영향을 미칠 수 있다.

데모를 위해 이용된 울펜스테인 게임 레벨은 오리지널 형태에서 300,000개의 트라이앵글로 구성되었다. 더 많은 기하학 구조가 가능하다는 것을 보여주기 위해서, 두 오브젝트가 과도하게 높은 폴리곤 디테일을 사용하는 것으로 변경되었다. 첫 번째는 샹들리에 모델이었고, 약 1,000,000개의 트라이앵글로 이루어졌다. 두 번째는 300,000개의 트라이앵글로 이루어진 자동차 모델이었다.




좌 : 구식의 샹들리에 모델 우 : 새롭고 향상된 디테일이 있는 샹들리에 모델

이 모델을 변경하고, 트라이앵글 디버그 쉐도우로 렌더링을 한 퍼포먼스 효과는 단지 15-20%에 불과하였다. 콘텐츠 생산 파이프라인 측면에서, 이것은 이러한 고품질의 레졸루션 모델이 3D 모델링 소프트웨어에서 공제될 수 있고, 고통스럽게 모든 가능한 트라이앵글을 감소시키려는 아티스트의 노력 없이 게임에서 직접적으로 활용될 수 있으며, 세부적인 기학학적 구조를 속이기 위해 일반적인 맵과 같은 속임수를 사용할 필요가 없다는 것을 의미한다.

쉐이딩

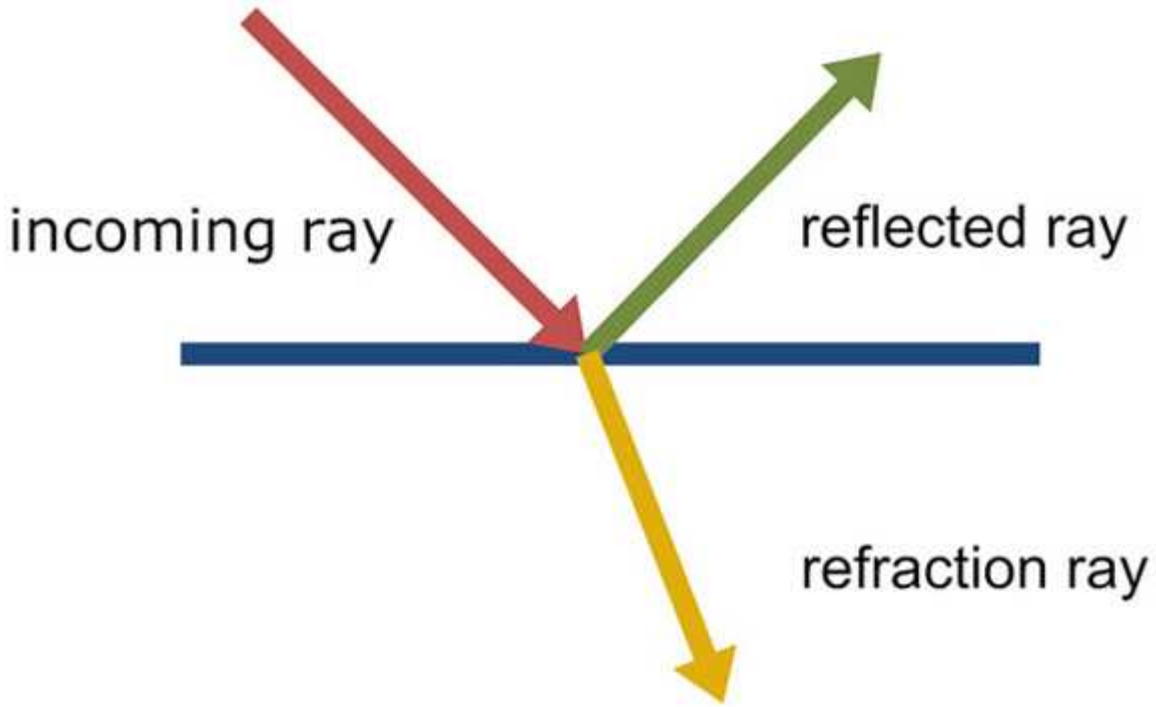
레이 트레이스에 능력에 대한 수요가 임의적임에도 불구하고, 레이 트레이싱은 쉐이딩에 흥미로운 영향을 미치고 있다.

반사를 이용하고 있는 대표적인 예는 자동차 모델에서 이루어지고 있다. 물체를 매우 반사적으로 만듦으로써, (사실 너무 반사적이라는 것이 현실적이지 않은 것 같지만, 실증의 목적에서 본다면 효과를 보이기에 좋음) 인근의 환경을 직면하도록 해 주었다 – 모든 업데이트가 정확하고 실시간으로 이루어진다.

레이 트레이서에서 이것이 계산되는 방식은 매우 간단하다. 당구대의 가장 자리에서 튕겨지는 당구공과 같이 반사되는 빛은 표현에 부딪혀 반사된다.

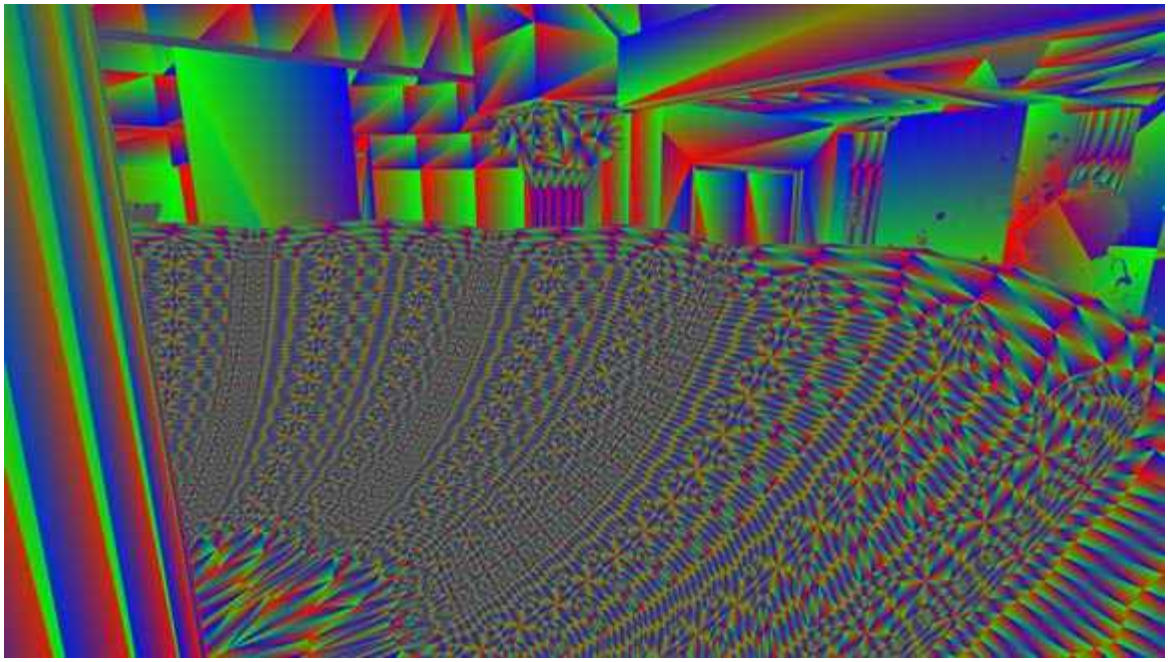
이 방법은 이와 같이 자동차 외관에 적용된다 :

또 다른 레이 트레이스의 좋은 효과는 올바른 굴절률과 같은 물리적 특성을 반영한 유리 쉐이더이다. 이전의 자동차 쉐이더와 비교해서, 이것은 한 지점에 부딪혀서 두 방향으로 나아가는 결과를 낳는다 : 위의 예와 같은 반사 빛과 굴절 빛이다.



샹들리에에 사용된 쉐이더는 이것처럼 보인다 :

레이 트레이스된 유리 쉐이더를 통해 보이는 배경의 왜곡



샹들리에 모델의 기학학적 구조의 디테일을 보여주는 것

게임 플레이에 영향을 미칠 수 있는 반사의 또 다른 예는 게임에 활용되는 스나이퍼 라이플 무기의 렌즈이다. 이 물체의 특징은 반사되는 코드를 몇 줄에 의해서 변경된다. 플레이어는 이제 그들 뒤에 무엇이 있는지를 볼 수 있으며(백미러와 같이) 빠르게 접근해 오는 적을 알아차릴 수 있게 될 것이다.

반사각을 이용한 스코프는 플레이어에게 그들 뒤에서 어떤 일이 벌어지고 있는지에 관한 정보를 제공한다.

또 다른 흥미로운 효과는 감시 카메라이다. 게임은 이미 이러한 종류의 것을 시행하고 있지만, 제한적인 용도에 그치고 있다. 때때로 다른 부분의 장면을 보여주는 스크린이 있는 방이 있다. 일부의 게임에서, 스크린 디스플레이의 지역을 변경할 수 있는 버튼이 있기도 하다. 여기서는 여전히 좋은 퍼포먼스 수준을 유지하면서 동시에 많은 수의 다른 스크린을 가질 수 있는 가능성을 제시한다.

좌 : 12개의 다른 부분을 보여주는 감시 카메라 우 : 쉽게 행해지는 반복 효과

방법은 간단하다. 각각의 스크린은 정확한 하나의 망점을 가지고 하나의 시각 방향을 가지게 된다. 빛이 이 화면에 닿으면, 망점이 그 지점에 부가된다. 이것으로부터, 또 다른 빛이 규정된 구체적인 방향으로 쉽게 따라가게 될 것이다. 이것이 색깔로 돌아오게 될 또 다른 표면을 스치게 될 것이다. 이 효과는 카메라 포털 효과와 관련이 있다. *Quake Wars*: Ray Traced project에서 이전에 시연되었다)그러나 TV 스크린에서의 효과와 달라서, 디스플레이 된 그림은 플레이어의 시각 각도와 똑 같은 독립체로 남아 있게 된다. 반면 포털에서는 플레이어가 앞에서 보는 대신에 포털의 안에서 보게 됨으로써 다른 정보를 얻을 수 있다. 포털 효과 때문에 발생하는 것으로, TV 스크린이 가지고 있는 고정된 방향과 비교해서, 빛이 전달되어 지는 방향이 고려되어야 한다.

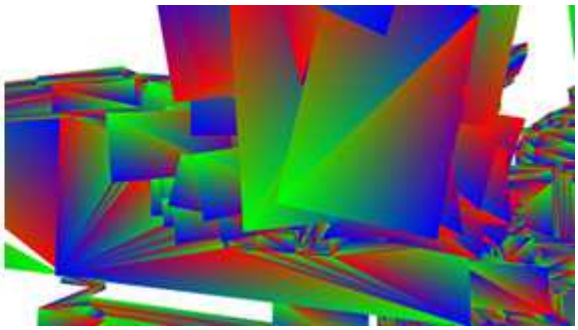
입자 시스템

요즘 게임은 연기, 불, 스파크, 잔해, 움직이는 덩어리의 안개 등등과 같은 효과를 디스플레이 하기 위해서 입자 시스템을 심하게 이용한다. 초기에 입자는 매우 작고, 한가지의 색깔로 된 사각형 이었다.

Quake (I)에서의 흐르는 용암 입자 트레일

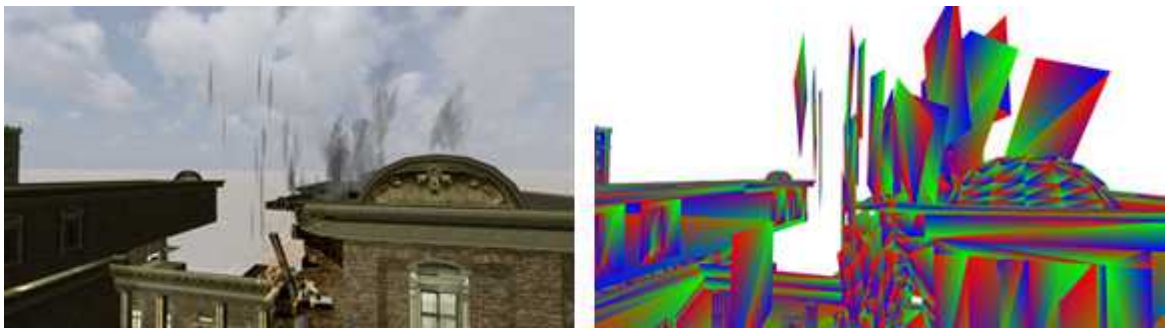
오늘날, 게임의 입자 효과는 전형적으로 두 개의 트라이앵글로 구성된 "쿼드"의 활용을 수반한다. 이것에 이용되는 쉐이더는 다양한 텍스쳐를 혼합할 수 있고, 다른 블렌딩 방법, 꼭짓점 색깔 및 프로그래밍할 수 있는 모든 마술 같은 쉐이더를 활용할 수 있다.

게임에서의 연기와 불의 실행은 많은 쿼드의 층을 가진 입자 시스템을 활용하고 있으며, 이를 통해 질량감을 가지는 효과를 시각적으로 만들 수 있다.



입자를 이용한 연기와 불. 좌 : 쉐이드 뷰 우 : 트라이앵글 디버그 뷰

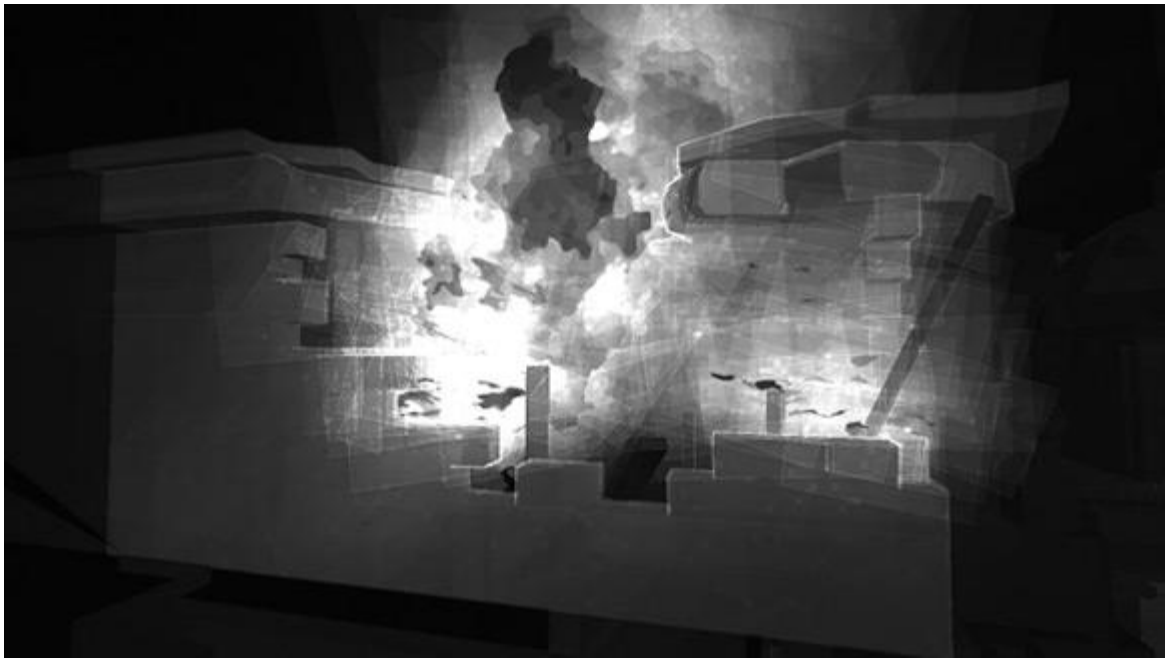
일반적으로 이러한 입자의 많은 층은 플레이어의 눈을 향해 정열된다. 디버깅 목적으로, 이 셋업은 고정될 수 있고, 레이어 양에 관한 좀더 세부적인 것을 드러내는 측면으로 볼 수 있다.



고정 입자 뷰 측면에서 보는 디버그 뷰 좌 : 쉐이드 뷰 우 : 트라잉앵글 디버그 뷰

레이어를 이용한 이러한 근사치는 래스터화를 이용하는 게임에서 개발되어졌으며, 이 알고리즘을 빠른 방법으로 계산할 수 있었기 때문이다. 우리는 이 데모와 같은 방식으로 실행하였으나 레이-트레이싱 게임에 맞게 질량을 보여주는 다른 알고리즘을 사용해야 한다는 것을 추천한다.

이러한 층 효과를 올바르게 디스플레이하기 위하 위한 레이 트레이서를 위해서, 빛은 첫 번째 표현에 부딪혀야 한다. 쉐이더 프로그램은 텍스쳐 색인 후에 색으로 되돌아오도록 하며, 일부의 투명율을 규정한다. 여기서, 빛은 다음 표면에 부딪힐 때 까지 이 단계를 반복하기를 계속해야만 한다. 우리의 테스트에서, 우리는 합당한 디스플레이를 보장하기 위해 50 뎁스의 반복이라는 힘든 케이스를 선택하였다. 이것은 매우 퍼포먼스 집중적인 테스트가 일 수 있으며, 사실 레이 트레이서를 이용하는 래스터 최적회된 방식에서 이러한 입자 렌더링은 데모 버전에서 가장 비싼 시각 효과라 할 수 있다.



디버그 뷰는 입자를 렌더링 할 때 뜨거운 지점의 퍼포먼스를 보여준다. 밝은 곳은 어두운 곳 보다 훨씬 집중되어 있다는 것을 의미한다.

모든 표면이 똑 같은 방향으로 나 있으며 쿼드라는 제한점을 가진다면, 이것을 렌더링할 때 적용할 수 있는 최적화가 있다. 또한 다르고 잠재적으로 더 빠른 렌더링 방법이 이미지를 사후 블렌드 할 때 이용될 수 있다.

레이어링을 통해서 효과를 시뮬레이팅 하지 않고, 실제의 질량감을 가진 데이터를 가지도록 하는 기술이 향후 개발될 것이다. 가령, 연기는 자연스럽게 공기중으로 흩어져야 하고 장애물에 따라 변경되어야 한다. 레이 트레이싱을 이용하면, 연기 구름의 내부의 밝은 값은 환경에 적합하게 적용될 수 있다.

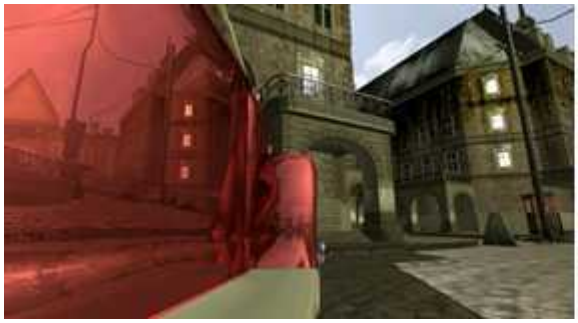
FumeFX를 이용한 시뮬레이트 된 연기. 작가 : Sam Khorshid

Video: http://www.afterworks.com/FumeFX/anims/SmokeCollection.wmv

퍼포먼스

클라우드 기반의 환경에서 어떻게 수행되는지를 보여주는 예가 있다.

78 fps 대 53 fps




66 fps 대 70 fps


62 fps 대 38 fps

지연

네트워크에 관해 가장 일반적으로 사용되는 두가지 용어가 대역폭과 지연이다 :

대역폭 : 주어진 시간에 특정 자원을 특정 목적지까지 네트워크를 통해 이동시킬 수 있는 데이터 양. 예를 들어 초당 12 Mbits는 가정용 인터넷 연결로 사용되고, 초당 1000 Mbits는 Gigabit Ethernet에 해당한다. 이것은 종종 고속도로와 비교되고 얼마나 많은 길이 있는지에 비유된다. 샌프란시스코와 로스엔젤레스 사이에 1000개의 길을 가진 고속도로가 있다고 하더라도, 차 한대는 1000배 빠른 속도로 운전할 수 없다.

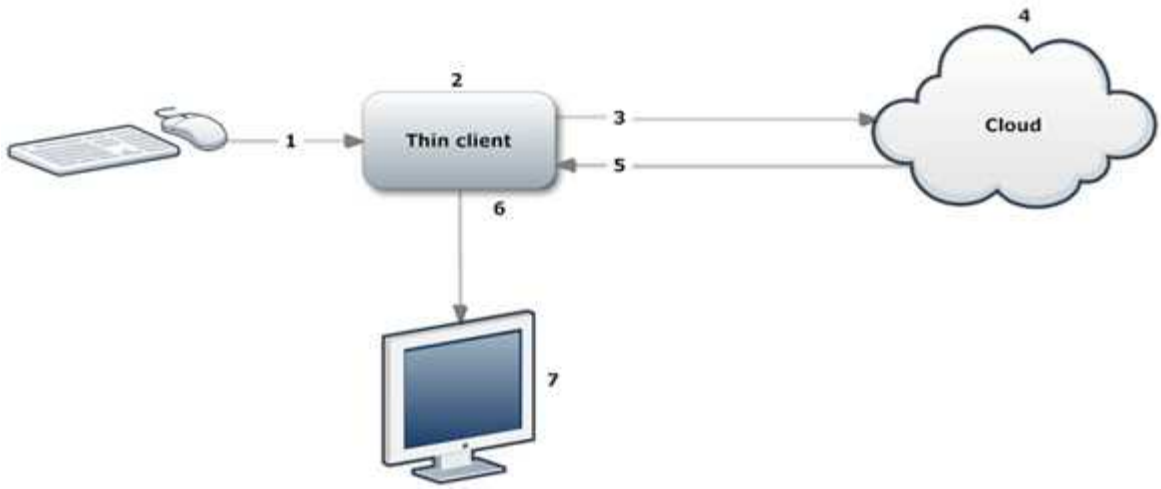
네트워크 지연 : 자원을 목적지로 네트워크 패킷으로 보낼 때 발생하는 시간 지연을 의미한다. 지연은 클라우드 기반의 게임 경험에 큰 영향을 미친다. 한 기계에서 다른 기계로의 패킷 지연은 ping 명령을 사용하여 테스트할 수 있다.

사용 가능한 대역폭이 제한되어 있지 않다고 가정할 경우, 지연은 빛의 속도라는 물리적 제한점에서 기인하며, 또한 네트워크 패킷이 하나의 물리적인 연결에서 또 다른 연결(라우터,

스위치 등)을 거치는 동한 다양한 홉으로부터 지연된다. 또한 운영 시스템과 최종 어플리케이션까지 거쳐야 하는 여러 단계를 통해서도 지연이 발생한다.

대역폭은 클라우드 기반의 환경에서 플레이어가 수신 받는 이미지의 품질에 영향을 미칠 것이다. 인터넷 연결 대역폭은 지난 세월 동안 증가하고 있고, 계속 증가할 것으로 기대되고 있다.

(전체) 지연은 마우스를 움직여서 스크린 상에 업데이트된 것을 볼 때까지 걸리는 시간이 얼마나 될지에 영향을 미칠 것이다. 첫 단계는 네트워크 지연에 국한된 문제일 수 있지만, 더 좋은 게임 경험을 제공하기 위해 조정되어 질 수 있는 상황에 기여하는 많은 다른 요소가 있다. 다은 단계를 따라서, 인풋 콘트롤러에서 스크린 업데이트까지 과정을 검토하고, 60 Hz의 목표 프레임 비율이 구성되는 것을 가정해 보겠다. 단순히 17 ms 정도 걸릴 것이라는 것에 반해, 이것은 각각의 프레임이 16 ms 걸린다는 것을 의미한다. 왕복(편도 40 ms) 80 ms의 네트워크 지연이 가정될 것이다.



인풋 콘트롤러에서 업데이트까지의 과정

단계 1 (인풋 콘트롤러) : Microsoft의 최신 운영 시스템에서, USB로 연결된 디바이스는 기본값 125Hz 모드 이다. 그러므로, 최악의 경우 이것으로부터 8 ms 지연이 발생할 수 있다. 이 비율을 가령 1000Hz의 비율로 바꿀 수 있는 특정 마우스 드라이버나 기술이 있다. 무선 키보드와

마우스는 더 높은 지연을 가지고 있는데, 랙을 감소시키기 위해 피해야만 한다.

단계 2 : 인풋 콘트롤러를 통한 움직임이 이루어진 다음, 게임 클라이언트가 이러한 움직임을 평가하는 코드를 받을 때 까지 짧은 시간이 소요될지 모른다. 예를 들어, 클라이언트는 (싱글 쓰레드 게임 클라이언트 버전에서, 최신의 렌더링 프레임을 받거나 그래픽 카드에서 이것을 업로드 하기 때문에 발생할 수 있음) 다른 계산 때문에 혼잡할지 모른다. 최대의 지연은 거의 한 프레임(17ms) 정도가 될 것이다. 게임 클라이언트에 멀티플 쓰레드를 이용함으로써, 이 랙은 감소될 수 있다. 그러나 언제 언데이트가 서버에 보내져야 할지에 대한 부가적인 싱크로나이제이션이 필요할지 모른다.

단계 3 : 모든 움직임을 평가 한 다음, 업데이트 된 플레이어 포지션과 같은 데이터를 가진 네크워크 패킷은 서버에 보내질 것이고 40ms를 필요로 한다. 일반적으로, 낮은 네트워크 지연에서, 적은 수의 홉에 이를 때까지 충분히 서버를 닫아 놓는 것이 도움이 된다. 일부의 라우터는 온라인 게임 네트워크 패킷을 최우선적으로 다루도록 특수하게 최적화 되어 있기도 하다. 이를 통해 클라이언트 디바이스에서 인터넷 연결 라우터까지 첫번째 홉을 도울 수 있을 것이다.

단계 4 : 이제 서버는 업데이트 된 정보를 받았다. 이전 프레임 렌더링을 마쳤다고 가정하고, 새로운 데이터가 활용될 수 있다. 그러나 게임은 종종 멀티플 프레임이 운행 중에 렌더링 되고 있는 장소를 이용하는 방법을 채택한다. 이것의 장점은 고 품질의 더 부드러운 퍼포먼스를 낼 수 있게 해 주는 것인데, 렌더링과 장면 업데이트를 위한 계산이 항상 충분히 가용할 만큼 있기 때문에 절대 멈추지 않기 때문이다. 그래픽 드라이버의 일반적인 기본 세팅은 세 프레임의 사전 렌더링 까지를 가능하게 해 준다.

이 값이 낮아지지 않는다면, 잠재적으로 50ms의 지연이 부가된다. 개별 렌더링 프레임이 있는 멀트플 기기 환경에서, 부가적인 지연이 발생한다. 앞서 언급했듯이, 타일 기반의 접근법은 이 오버세드를 피하기 위해 사용되어 진다. 일단 프레임이 렌더링 되고 나면, 주 메모리에 불러와 진다. 이미지는 디바이스에 직접적으로 압축되어 질 수 있고, 또는 나중에 CPU에 의해 압축되어 질 수도 있다. 서버가 항상 강력하기 때문에 압축은 (1280x720일 경우 DXT1으로 약

2mx 정도) 적은 프레임 타임을 소비한다.

단계 5 : 압축된 프레임은 클라이언트에 보내지고, 40ms를 소비한다.

단계 6 : 클라인트로 돌아와서, 압축된 프레임은 디스플레이된다. 압축된 이미지를 직접적으로 그래픽 카드에(예를 들어, DXT1 텍스쳐를 지원하는 카드) 업로드 되는 방법이 있을 수 있고, 또는 수동적으로 압축 해제 되도록 해야 한다. 클라이언트가 대체로 충분한 마력을 가지고 있지 않기 때문에, 이 단계는 하나의 프레임에 의해 지연될 수 있다 : 클라이언트는 이미 네트워크를 통해 다음 프레임을 받는 동안, 이전 프레임의 압축 해제를 마칠 수 있다. 이것은 또 다른 17ms의 지연을 발생시킨다. 따라서 작은 디바이스를 위한 최적화가 필요해질 수 있다. 클라이언트의 주 메모리에서 그래픽 카드로 색 데이터를 업로드 하는 것은 일정 정도의 시간이 소요되게 한다. 17ms 정도의 지연이 발생할 것이다. 또한 예를 들어 Microsoft DirectX를 통해 풀 스크린 상에 쿼드로 디스플레이 하고 텍스쳐를 업로드 해야 하는 그림인 경우에, 클라이언트에서 사전 렌더링되는 프레임의 최대치를 위한 그래픽 카드 세팅은 여분의 지연을 막을 수 있다.

단계 7 : 끝으로, 그래픽 카드를 떠나는 그림이 스크린 상에 디스플레이 되어져야만 한다. 최근의 대부분의 플랫 스크린 모니터는 60Hz로만 디스플레이 할 수 있는 제약이 있고, 올바른 싱크로나이제이션을 위한 지연이 발생할 수 있기 때문에, 인풋 랙이라 불리는 중요한 또 다른 요소가 있다 : 이것은 시그널을 모니터에 보내는 것과 스크린 상에 실제 내용물을 보는 것 사이에 발생하는 시간 차이를 말한다. 이 차이는 플랫 스크린 모니터가 처리하는 시그널 처리에 의해 발생한다. 가령, 콘트라스트와 컬러 조정, 고스팅에 의해 색 조절 및 블러링을 줄이도록 이미지를 처리하거나 몇 프레임 앞서서 디스플레이 될 것을 읽게 되는 "overdrive" 방법과 같은 일반적이지 않은 해결책으로 디스플레이 끼워 넣기와 같은 것이 될 수 있다. 0에 가까운 것에서 45ms까지 인풋 랙을 측정한 Prad.de에서 이것에 관한 세부적인 테스트를 가지고 있다. 이러한 값은 게이밍 모니터를 위한 그레이 투 그레이로부터 2ms와 같이 모니터 제조사에서 구체화하는 픽셀 응답 시간과 혼돈되어서는 안된다.

그러므로 총 지연 시간은 네트워크 지연에만 국한된 것이 아니며, 인풋 콘트롤러와 모니터 및 버퍼링과 관련된 실행 소프트웨어 선택 까지에 연결되어 있다. 100ms 지연 차이까지를 발생할 수 있는 시나리오가 가능하다. 네트워크 지연과는 상관없이 말이다.

	시나리오 예 1	시나리오 예 2
단계 1 (인풋 콘트롤러)	8	1
단계 2 (클라이언트)	17	17
단계 3 (네트워크)	40	40
단계 4 (클라우드)	50 (트리플 버퍼)	17 (싱글 버퍼)
단계 5 (네트워크)	40	40
단계 6 (클라이언트)	17 + 17	17
단계 7 (모니터)	45	0
합계	234	132

다른 하드웨어와 소프트웨어 셋업은 총 지연 시간을 크게 변동시킬 수 있다.

결과 논의

마지막 부분에서 레이 트레이싱이 게임에 다양한 새롭고 흥미로운 영향을 미칠 수 있다는 것을 설명하였다. Knights Ferry Intel 코드 네임 개발 플랫폼을 활용한 기기를 가진 클라우드 기반의 게임 환경을 이용한 이 글을 진행하고 있는 동안, 고 프레임 비율의 레이 트레이싱 게임이 이미 만들어 지고 있다.

앞으로의 진보는 넷북 및 태블랫과 같은 더 작은 디바이스에서 조차 이용할 수 있도록 비디오 코덱을 최적화하여 만드는 것이다.

Gigabit Ethernet 환경 대신에, 스마트폰과 같은 휴대용 디바이스로 기술을 가져오도록 하는 무선 네트워크 최적화가 연구될 수 있다. 필요한 서버의 수를 줄이기 위해서, 하나의 기기 안에

멀티플 Knights Ferry PCIe 카드를 이용하는 지원방법 개발이 가능해 져야 한다. 이미지의 품질을 증가시키기 위해, 몇 가지 잘 알려진 HDR 블룸과 필드 뎁스와 같은 사후 처리 기술이 부가될 수 있다. 또한 Knights Ferry 플랫폼에서 높은 퍼포먼스를 가진 레이 트레이승을 위해 반 에일리어싱 방법에 대한 똑똑한 해결책이 연구될 수 있다.

클라우드 기반의 게임의 상업적 이용이 막 시작되고 있는 동안, 일부 게이머가 이러한 접근법에 대한 장점을 선호하게 되고, 이 방법을 게임 이용의 주요 방법으로 삼게 될 수 있다는 시나리오를 그려볼 수 있다. 또한 또 다른 그룹의 게이머가 현재 모델에 머무르는 것을 좋아하여, 모든 게임이 게이머의 기계에서 렌더링 되거나 저장될 수 있게 될 수도 있다고 가정하는 것도 가능해 질지 모른다.

어떤 것이 더 오래 지속될지와 상관없이, 산업은 놀라운 속도로 변하고 있으며 게임 그래픽, 클라우드 기반 게임, 가장 중요한 게임 경험이 얼마나 빠른 속도로 진화하고 있는지를 지켜보는 일은 흥미로울 것이다.