

클라이언트 서버 기반 모바일 볼륨 가시화 시스템

이용규^o 계획원*

한성대학교 정보시스템공학과

teriusbin@naver.com

kuei@hansung.ac.kr

Mobile Volume Rendering System for Client-Server Environment

Woongkyu Lee^o Heewon Kye*

Dept. of Information Systems Eng, Hansung University

요 약

본 연구는 클라이언트 서버 기반의 볼륨 가시화 시스템에 대해 설명한다. 소형 병원에서의 볼륨 가시화 시스템은 소수의 사용자만이 동시에 사용한다는 점에 착안하여, 단일 GPU를 장착한 PC를 렌더링 서버로 사용하고 클라이언트는 현재 대중적으로 사용하는 스마트 폰과 같은 안드로이드 기반의 모바일 장비를 사용하였다. 사용자가 클라이언트 응용 프로그램을 이용하여 렌더링 요청을 하면, 서버는 GPU를 사용하여 볼륨 가시화를 수행한다. 렌더링 영상은 서버에서 JPEG나 PNG 형식으로 압축하는데, 네트워크 전송량을 감소시켜 가시화 속도를 향상시킬 수 있다. 추가적으로 사용자가 터치 스크린을 드래그 하는 경우 반응시간을 향상하기 위해, 사용자가 발생하는 일부의 이벤트를 제거하며 서버는 제거된 이벤트를 보간을 통해 보상하는 방법을 제안한다. 그 결과로 제안 시스템은 5명의 동시 사용자에게 대해 GPU를 장착한 단일 상용 하드웨어로 실시간 볼륨 가시화가 가능하였다.

Abstract

In this paper, we explain a volume rendering system for client-server environment. A single GPU-equipped PC works as a server which is based on the ideas that only a few concurrent users use a volume rendering system in a small hospital. As the clients, we used Android mobile devices such as smart phones. User events are transformed to rendering requests by the client application. When the server receives a rendering request, it renders the volume using the GPU. The rendered image is compressed to JPEG or PNG format so that we can save network bandwidth and reduce transfer time. In addition, we perform an event pruning method while a user is dragging the touch to enhance latency. The server compensates the pruning by interpolating the touch positions. As the result, real-time volume rendering is possible for 5 concurrent users on single GPU-equipped commodity hardware.

키워드: 볼륨 렌더링, 의료 영상, 모바일, 클라이언트/서버, GPU

Keywords: Volume rendering, Medical image visualization, Mobile, Client/Server, GPU

1. 서론

볼륨 가시화는 3차원 배열 형식으로 구성된 볼륨 데이터를 가시화 하는 방법[1,2]으로 주로 의학용 혹은 공학용 가시화에 사용되고 있다. 볼륨 데이터는 일반적으로 복셀(voxel)이라고 부르는 스칼라(scalar)값의 3차원 형태로 구성되어 있는데, 볼륨 데이터의 크기는 매우 크기 때문에 영상을 생성하는데 오

랜 시간이 소요된다. 이러한 문제를 해결하기 위해 병렬 프로그래밍을 통한 가속화 방법들이 연구되고 있다.

최근 비용 및 편익 관점에서 클라이언트 서버 모델의 가시화 시스템이 주목 받고 있다. 여러 사용자에게 각각 가시화 시스템을 제공하면, 하드웨어 설치비용 및 시스템 유지보수 비용이 많이 발생하기 때문에 비율을 적이다. 하지만 클라이언트 서버 모델의 경우 하나의 서버에 다수의 사용자가 접속하기 때문에 비용 및 유지보수 측면에서 효율적이다. 단일 서버

*corresponding author: Heewon Kye/Hansung University(kuei@hansung.ac.kr)

에 수백 혹은 수천 명이 접속하여 서비스를 제공하는 게임이나 웹 서버와 달리 볼륨 가시화 시스템은 소수의 인원이 동시에 사용하게 된다.

한편 스마트 폰과 같은 모바일 기기의 대중화로 기존의 응용 프로그램이 모바일(mobile) 기기에서 구현되고 있다. 표준 그래픽스 라이브러리인 OpenGL ES의 보급 확대를 기반으로 모바일 기기를 이용한 그래픽스 연구가 진행되고 있다. 하지만 대용량 3차원 데이터로부터 출력영상을 생성하는 볼륨 가시화의 경우, 많은 계산 비용과 충분한 메모리 공간을 필요로 하므로 현재의 모바일 기기에서 고속의 볼륨 가시화는 무리가 있다[3].

본 연구는 클라이언트-서버 모델의 볼륨 가시화 시스템을 구축하는 것을 목표로 하였다. GPU(graphic processing unit)로 구성된 볼륨 렌더링 서버를 이용하여 여러 사용자의 가시화 요청에 대응하고, 안드로이드(android) 기반의 모바일 클라이언트에서 동작하는 응용 프로그램을 개발하였다. 따라서 모바일 기기의 연산 장치 및 메모리 크기의 한계를 극복하며 실시간으로 볼륨가시화가 수행 가능하다. 안드로이드 기반으로 제작된 클라이언트 프로그램은 다수 하드웨어에서 변경 없이 동작 가능하다.

본 연구는 소규모 병원에서 단일 서버를 구축하고, 환자 진단을 수행하거나 환자에게 진단을 설명할 때 이동성이 좋은 태블릿이나 스마트 폰을 이용하여 무선 네트워크를 통해 실시간으로 가시화 하는 환경에 매우 적합하다. 논문의 구성은 다음과 같다. 2장에서는 본 시스템과 관련된 기존의 연구들에 대해 설명한다. 3장에서는 시스템의 구체적 설계에 대해 자세히 설명하고 4장에서는 실험 결과를 보인다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

클라이언트-서버 기반 볼륨 가시화에서 클라이언트로는 보통 PC가 사용되었다. 성능이 낮은 클라이언트 PC는 고성능 가시화 서버에 접속하여 대용량 볼륨 데이터를 효과적으로 가시화한다. 예를 들어 ScanView 시스템[4]은 데스크톱 환경에서 고속의 가시화 영상을 생성할 수 있다. 또한 고화질의 영상을 생성하기 원한다면 서버에 요청하여 새로운 영상을 받을 수 있도록 구현되어 있다. 이와 더불어 VR2[5]의 경우 볼륨 가시화 응용 프로그램에 가상현실을 접목시킨 시스템으로, 서버에서 볼륨 가시화를 수행하고 PC에서 가상현실 하드웨어를

이용해 볼륨 가시화를 수행할 수 있다. 볼륨 가시화의 연산 부하가 높다 보니 다중 사용자를 지원하는 서버는 최근여야 연구되기 시작하였다[6]. 해당 시스템은 사용자의 세션(session)을 관리하는 GSM(global session manager)과 가시화 연산을 관리하는 VSM(view session manager), 마지막으로 하드웨어 부하를 제어하는 RRLB(rendering resource load balancer)로 구성되어 있다. 본 논문에서 제안하는 시스템도 클라이언트-서버 모델을 기반으로 개발된 다중사용자 지원 시스템이라는 공통점이 있다. 다만, 본 논문의 클라이언트 환경은 휴대가 간편한 모바일 환경으로 이라는 점, 서버는 GPU 시스템을 이용하여 설계되었다는 점에서 차이가 있다.

한편 모바일 장비의 성능이 향상되면서 모바일 하드웨어를 직접 이용한 볼륨 가시화 연구[3]도 진행되고 있다. 이 연구는 ios의 모바일 GPU를 사용하기 위해 OpenGL ES의 셰이더(shader) 기능을 사용하여 볼륨 가시화 연산을 수행한다. 이를 통해 모바일 기기에서 볼륨 가시화가 가능하다. 또 다른 방법으로는 모바일 환경에서 웹 브라우저(web browsers)를 통해 렌더링을 수행하는 연구[7]가 있다. 모바일 기기의 GPU와 WebGL을 이용하여 볼륨 가시화 연산을 수행한다. 그리고 이에 대한 결과 영상을 HTML5의 캔버스(canvas)를 이용하여 웹 브라우저에 출력한다. 그러나 모바일 장비는 연산 자원이 부족하기 때문에, 이러한 연구들은 실용적인 용량의 데이터를 실시간으로 가시화할 수 없다는 한계가 있다.

마지막으로 본 연구와 유사하게 모바일 기기를 클라이언트로 삼고 렌더링 서버를 이용하여 볼륨 가시화를 수행하는 연구도 존재한다. 이 연구[8]는 PDA환경에서 발생한 사용자 이벤트를 서버에 전송하고, 서버에서 이벤트에 맞는 가시화 연산을 분산된 서버를 이용하여 병렬적으로 수행한다. 이후 중앙 서버에서는 가시화 연산 결과인 2차원 영상을 취합하고 이를 인코딩(encoding)하여 사용자에게 송신한다. 또 다른 방법으로는 기존연구[8]의 사용자 반응성을 개선하기 위해 스트리밍 서버를 별도로 구축하는 연구[9]도 있다. 해당 연구에서는 네트워크의 병목현상을 개선하기 위해 MPEG 동영상 형태로 인코딩하여 사용자에게 결과 영상을 전송한다. 실시간 스트리밍을 위하여 비디오테이터 실시간 전송을 위한 프로토콜인 MPEG-TS(MPEG transport stream)를 사용한다.

본 논문에서 제안하는 시스템도 클라이언트-서버 모델을 기반으로 개발되었지만, 안드로이드와 OpenGL ES 표준을 사용하여, 구동 가능한 클라이언트의 범위가 매우 넓다. 또한 멀티터치와 같은 편리한 안드로이드 기반의 UI를 제공하여 조작성

이 우수하다. 그리고 기존 시스템과 달리 단일 GPU 서버 환경에서 다중 사용자에 대해 대응 가능하다.

예를 들어 의료영상 시스템을 사용하는 소규모 병원에서, 한 대의 렌더링 서버만을 구축하여 두고, 여러 명의 환자에게 각기 결과를 보여주고자 할 때는, 태블릿이나 스마트폰을 이용하여 편리하게 진단을 보조할 수 있다.

3. 시스템 구성

본 연구에서 제안하는 시스템은 그림 1과 같이 모바일 클라이언트와 가시화 서버로 구성되어 있다. 모바일 클라이언트는 터치와 버튼 선택과 같은 사용자 이벤트를 적당한 렌더링 명령으로 변환하고 볼륨 가시화 영상을 서버에 요청한다. 가시화 서버는 요청된 명령에 따라 병렬 볼륨 가시화 알고리즘을 GPU(graphic processing unit)를 이용해 수행한다. 이후 서버에서는 결과 영상에 대한 인코딩 과정을 거쳐, 압축 영상을 사용자에게 전송한다. 다시 모바일 클라이언트는 수신된 영상을 디코딩해 화면에 출력한다. 이를 통해 사용자는 다양한 효과의 가시화 영상을 실시간으로 제공받을 수 있다.

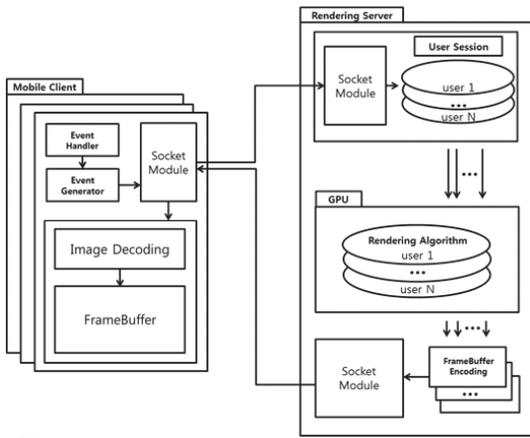


Figure 1. System architecture

3.1 사용자 입력

모바일 클라이언트는 그림 2와 같이 사용자 이벤트 입력을 고유한 렌더링 명령으로 변환하여 서버에 전송한다. 본 연구에서 관찰자 변환은 표준적인 안드로이드의 터치 이벤트 의해 제어된다. 사용자의 터치는 멀티 터치, 드래그를 포함하여 다양하게 수행될 수 있으며 이를 편리하게 확대, 회전 등의 관찰

자 변환으로 동작할 수 있도록 UI를 설계하였다.

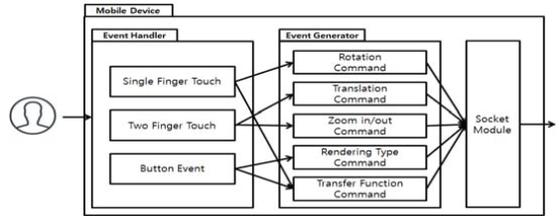


Figure 2. Client input

가장 자주 수행하는 회전 변환은 싱글 핑거 터치(single finger touch) 드래그를 이용하였고 드래그 방향에 따라 회전 방향을 계산하여 회전을 수행한다. 볼륨의 확대/축소 변환은 보통의 사진 편집 프로그램과 유사하게 두 핑거 터치의 핀치 줌(pinch zoom)이벤트의 정도에 따라 확대/축소 영상을 생성한다. 그리고 가끔 사용하는 평행 이동은 사용자의 두 핑거 터치(two finger touch) 드래그 이벤트 방향에 따라 이동 영상을 사용자에게 전달한다.

이 때 볼륨의 평행 이동과 확대/축소 기능을 구분하기 위해 화면 터치 시 터치 이벤트의 방향을 통해 두 이벤트를 구분한다. 예를 들어 두 핑거 터치 이벤트가 발생했을 경우 터치 이벤트의 이전 좌표와 현재 좌표의 방향을 이용하여 동일한 방향일 경우 평행이동, 반대 방향일 경우 확대/축소 기능이라 판단한다. 볼륨 가시화에서는 사용자가 볼륨 데이터의 일정 부분을 투명하게 또는 불투명하게 설정하여 원하는 부분만 선택적으로 관찰하는데, 이런 설정을 불투명도 전이함수 (transfer function, TF)라고 한다. 그림 3은 TF에 대한 UI 화면이다. TF 변환 이벤트 및 볼륨 가시화 타입 변화 이벤트는 버튼 이벤트를 이용하여 구현하였다. TF변환의 경우 사용자의 버튼 터치 위치에 따라 해당 TF에 맞는 영상을 사용자에게 전달한다. 그리고 버튼 선택을 통해 다양한 형태의 가시화 타입 변경이 가능하다.

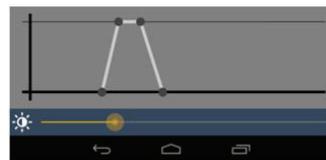


Figure 3. Transfer function

본 연구는 실시간 가시화를 수행하므로 반응속도(latency)가 매우 중요하다. 버튼의 선택과 같은 명령에 비해 관찰자 회전이나 확대 축소와 같은 드래그 터치는 매우 조밀하게 이벤트가 발생하는 특성이 있다. 각 이벤트에 대응하여 영상을 생성하게 될 경우, 서버는 사용자가 네트워크로 전송한 이벤트를 매번 해석하고 처리해야 한다. 따라서 잦은 신호 전송은 반응 속도 저하의 원인이 된다.

따라서 본 연구는 사용자가 발생한 이벤트를 서버에 모두 전송하는 것이 아니라 일부만 전송하여 이벤트의 전송횟수를 줄인다. 예를 들어 사용자가 드래그를 수행하는 동안 터치 인식 횟수가 100회 발생했을 경우 50회의 이벤트만을 서버에 전송한다. 즉 한 번 걸러 하나씩의 이벤트만을 사용자에게 전송하는 것이다. 서버는 보간을 통해 손실된 중간 이벤트를 스스로 파악하여 추가 영상을 생성한다. 앞서 설명한 바와 같이 드래그 터치는 매우 조밀하게 발생되므로 보간을 이용하더라도 드래그를 수행중인 사용자는 차이를 눈치채기 어렵다. 이 방법은 네트워크 상에서 사용자가 발생하는 이벤트를 해석하는 시간을 경감시켜 가시화 속도가 향상된다.

3.2 가시화 서버

3.2.1 볼륨 가시화

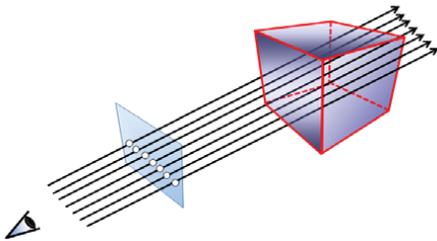


Figure 4. Volume ray-casting

그림 4는 광선 투사법(volume ray-casting)으로 볼륨 가시화 기법 중 높은 화질의 영상을 제공하는 대표적인 기법이다 [1,2,10]. 광선 투사법은 결과 영상의 각 화소 색상을 결정하기 위해 모든 화소별로 가상의 광선을 3차원 볼륨 데이터에 발사한다. 그리고 광선 진행 중 교차하는 모든 복셀의 밝기(또는 밀도) 값을 계산하고 TF를 이용하여 색상을 결정해서 누적한다.

이때 누적된 색상이 화소의 색상이 된다[11]. GPU 기반의 병렬 볼륨 가시화는 수백 개의 그래픽스 프로세서를 이용하여 동시에 수백 개의 광선을 투사함으로써 고속의 볼륨 가시화가 가능하다. 광선 투사법의 각 광선들은 서로의 진행방향과 색상결정이 독립적이므로 각 프로세서에 서로 다른 광선을 할당하여 병렬 연산을 수행하면 효과적이다.

3.2.2 CUDA를 이용한 GPU 기반 볼륨 가시화

GPU를 이용한 일반적인 프로그래밍 방법이 GPGPU이며, 그 종류로는 OpenCL[12]과 CUDA[13]등이 있다. OpenCL의 경우 공개 병렬 프로그래밍 표준으로, Khronos group이 유지 및 관리하고 있으며, CUDA는 nVidia에서 개발하여 nVidia GPU에서만 동작하는 프로그래밍 모델이다. 본 논문에서 제안하는 시스템의 경우 CUDA를 이용하여 가시화 서버를 구축하였다. CUDA를 이용하여 광선 투사법을 구현할 경우, 보통 하나의 스레드가 하나의 광선을 담당하게 된다. 이를 통해 광선들은 서로의 진행방향과 색상 결정이 독립적이므로 병렬 연산에 효과적이다.

본 연구에서는 가시화 알고리즘 가속화를 위해 텍스처 메모리(texture memory)를 사용한다. 텍스처 메모리는 GPU에서 제공하는 캐시 읽기를 지원하는 읽기전용 메모리이다. 텍스처 메모리에 저장된 데이터는 캐시에서 재사용되기 때문에 빠른 성능을 발휘할 수 있다. 본 연구는 3차원 볼륨 데이터를 텍스처 메모리에 저장하여 빈번하게 일어나는 메모리 참조 시간을 감소시켰다.

그리고 볼륨 가시화 속도를 가속하기 위해 빈 공간 도약(empty space leaping)과 조기 광선 종료(early ray termination) 기법을 사용하였다[14-17]. 빈 공간 도약 기법은 출력 영상에 반영되지 않을 영역을 미리 추출하여 건너내는 방법이다. 본 연구에서는 전 처리 단계에서 볼륨 데이터를 블록으로 나누어 각 영역의 최댓값 및 최솟값을 검색하여 CUDA 텍스처 메모리에 적재하여 사용한다. 이를 가시화 수행 중에 참조하여 광선이 진행 도중 마주친 블록이 투명한지를 판단하고 투명한 블록이면 건너뛰어 가속화 한다. 특히, 볼륨 텍스처의 경우 샘플링의 선형 보간이 필요하지만, 블록 데이터의 경우 선형 보간을 수행하면 오류가 발생하므로, 샘플러 설정을 별도로 관리하여 최근접(nearest neighbor) 보간으로 설정해야 한다. 그리고 조기 광선 종료 기법은 각 광선이 누적해 나가는 색상의 불투명도가 매우 커지면 더 이상 광선의 진행이 불필요 하다고

판단해 연산을 중지하는 방법이다.

한편 볼륨 가시화 영상의 화질을 향상시키기 위해 선-적분 볼륨 가시화[18]를 사용하였다. 선-적분 볼륨 가시화는 볼륨 광선 적분의 화질을 효과적으로 향상하는 가시화 방법이다. 이 방법은 가시화 이전에 광선의 진행 방향에 위치한 복셀의 스칼라 값이 선형성을 띠고 있다고 가정하고, 광선의 양 끝 점의 값을 이용하여, 광선 사이의 색과 투명도의 적분 값을 2차원 참조표(lookup table)로 미리 적분하여 구성해 놓는다. 이때 적분 참조표는 4096×4096 크기의 2차원 테이블이다.

본 연구에서는 사용자가 TF를 변경할 때 마다 적분 참조표를 생성한다. 적분 참조표는 GPU를 사용하여 실시간 속도로 병렬 생성이 가능하다. 이 때 생성하는 CUDA 스레드(thread)의 개수는 적분 참조표 크기와 동일하게 4096×4096으로 설계하였다. 블록 당 스레드의 개수는 32×32, 블록의 개수는 128×128로 지정하였다. 생성된 참조표는 텍스처 메모리에 적재하고 가시화 연산 중에 이를 참조하여 볼륨에 대한 색상과 투명도를 얻는데 사용한다.

3.2.3 영상 압축

볼륨 가시화 알고리즘을 통해 결과영상이 생성되고 이 영상은 클라이언트로 전송된다. 가시화 영상은 실시간으로 클라이언트에게 전송되기 때문에, 전송 데이터가 많고 오랜 시간이 소요된다. 사용자의 화면 출력 속도는 상대적으로 빠르기 때문에 네트워크에서 병목현상이 발생하게 된다. 이 문제를 해결하기 위해 본 연구는 결과 영상을 압축하여 전송하는 방법을 선택하였다.

사용자가 연속적으로 이벤트를 발생시켜 렌더링을 요청하는 경우, 빠르게 대응하기 위해 네트워크 대역을 크게 줄이는 손실 압축 방법인 JPEG(joint photographic coding experts group) 인코딩을 사용하였다. 서버에서 영상을 압축하는데 약간의 추가 시간이 소요되나 출력 영상의 크기가 감소되기 때문에 네트워크 상의 부하가 줄어들어 반응 속도 및 가시화 속도가 향상된다. 이때 손실 압축으로 인해 약간의 화질 손실이 발생할 수 있으나, 사용자가 관찰 방향을 바꾸거나 TF를 변경하는 등 다양한 명령을 연속적으로 수행하고 있는 상황을 가정하므로, 사용자 영상은 실시간으로 갱신되어 화질 저하를 크게 느끼지 못한다.

한편 렌더링 요청이 일시적으로 멈춘다면, 사용자는 자세한 관찰을 하고 있다고 판단할 수 있으며, 고화질의 영상이 필요

하게 된다. 이벤트 발생이 없는 경우 비 손실 압축 방법 중 하나인 PNG(portable network graphics) 인코딩을 사용하여 직전 JPEG 영상과 동일한 영상을 손실 없이 압축하여 추가로 영상을 전송한다. 이를 통해 사용자가 이벤트 발생을 멈추고 관찰을 하는 동안에는, 특별한 요청 없이 고화질의 영상을 제공할 수 있다.

3.3 영상출력

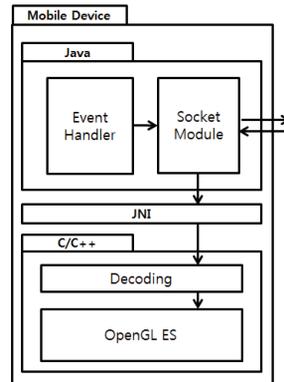


Figure 5. Client architecture

본 연구에서는 응용 프로그램의 영상 출력 성능 향상을 위해 JNI(java native interface)를 사용하였다. JNI는 C/C++같은 자바 이외의 언어로 만들어진 네이티브(native) 모듈과 자바 간의 상호작용을 할 수 있게 정의한 인터페이스를 의미한다. Java는 네이티브 프로그램에 비해 느리기 때문에 서버와 통신을 담당하는 모듈만을 Java를 통해 구현 했고 상대적으로 연산이 많은 디코딩과 화면 출력은 C++를 이용하여 구현하였다. 서버에서 수신된 영상은 압축된 영상이기 때문에 안드로이드 표준 API인 BitmapFactory[19]를 이용하여 JPEG 영상에 대한 디코딩(decoding) 과정을 수행한다. 이 후 디코딩된 영상은 OpenGL ES의 텍스처로 변환하여 화면에 출력된다.

4. 시스템 수행 결과

본 연구는 클라이언트 응용 프로그램 개발을 위해 Google ASUS Nexus7을 사용했고, OS(operating system)환경은 안드로이드 4.4.4 킷캣 환경에서 개발 하였다. 가시화 서버의 사양은 i7 CPU, 8GB RAM, Geforce GTX 970 GPU이며 OS환경은

ubuntu 12.04에서 개발 하였다. 소프트웨어 개발 환경은 IntelliJ 를 사용하였고 C++/java 언어를 사용하여 개발하였다. 병렬 볼륨 가시화 알고리즘 가속화를 위해 CUDA를 사용하였으며, JPEG 압축 성능 향상을 위해 SIMD명령어로 작성된 JPEG 라이브러리인 libjpeg-turbo[20]를 사용하였다.



Figure 6. Volume list page

그림 6은 모바일 클라이언트의 초기 구동 화면을 캡처한 것이다. 사용자가 가시화 서버에 볼륨 데이터를 업로드 하게 되면 서버는 기본 설정의 TF와 관찰 각도로 우선 렌더링을 수행하여 결과 영상을 썸네일(thumbnail)로 만들어 서버에 저장한다. 이후 사용자가 서버에 접속할 경우, 이 영상들을 사용자에게 전송하여 리스트 뷰(list view)형태로 가시화 한다. 사용자는 리스트 페이지를 통해 자신이 등록한 볼륨에 대해 미리 보기가 가능하며 터치 이벤트를 통해 볼륨에 대한 선택이나 볼륨 데이터의 수정이 가능하다.

그림 7은 다양한 가시화 결과를 모바일 클라이언트에서 캡처한 것이다. 좌측 상단 그림부터 시계 방향으로 512×512×1165 크기의 다리 데이터(leg data), 512×512×552 크기의 머리 데이터(head data), 256×256×225 크기의 머리 데이터(head data), 256×256×256 크기의 치아 데이터(teeth data)이다. 그림과 같이 화면 중앙에 볼륨 데이터가 가시화 되고 좌측 상단의 메뉴 버튼을 이용하여 다양한 효과에 대한 선택이 가능하다.



Figure 7. Rendering results
(Left top: Leg Data, Right top: Head Data, Left bottom: Head Data, Right bottom: Head Data)

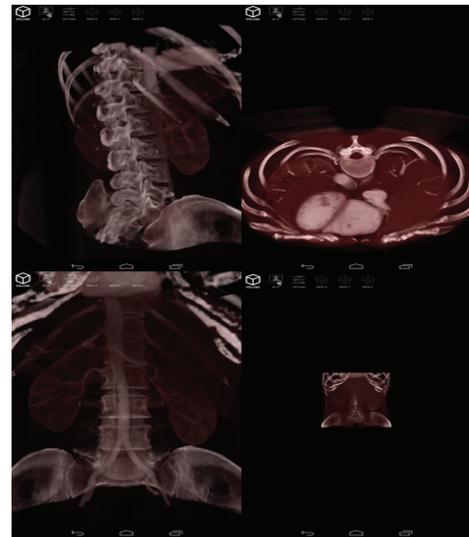


Figure 8. Geometric transformation by user events
(Left top: Rotation, Right top: Translation, Left bottom: Zoom in, Right bottom: Zoom out)

그림 8은 사용자의 터치 이벤트에 따른 기하변환 결과 영상이다. 사용자는 자세한 관찰을 위해 싱글 터치 드래그 이벤트를

를 통하여 볼륨 데이터의 회전이 가능하다. 또한 두 핑거 터치 드래그 이벤트를 통하여 평행 이동이 가능하다. 마찬가지로 두 핑거 터치의 핀치 줌(pinch zoom) 이벤트를 통하여 확대 축소가 가능하다.



Figure 9. Transfer function change by user events

그림 9는 TF 이벤트 변경에 따른 가시화 화면이다. 좌측 상단의 TF 버튼을 터치할 경우 TF UI 화면이 출력된다. 사용자는 TF UI의 꼭지점 및 선분을 터치하면서 다양한 형태의 볼륨 가시화가 가능하다. 그리고 하단의 밝기 조절 슬라이드 바를 이용하여 조명의 밝기를 조절할 수 있다.

그림 10의 좌측 상단 그림은 MIP (maximum intensity projection) 영상에 대한 출력 결과이다. MIP 영상은 광선의 진행 방향을 따라 가며 볼륨의 최대 밀도를 찾는 방법이다. 사용자는 좌측 상단의 MIP버튼 터치를 통하여 렌더링 타입을 Volume rendering 에서 MIP로 변경할 수 있다. 그림 8의 나머지 그림은 MPR(multiplanar reformatting)영상에 대한 출력 결과이다. MPR 영상은 볼륨 공간상에 임의의 평면을 정의하고 볼륨과 평면과의 교차점에서의 색상을 획득하는 방법이다. 마찬가지로 좌측 상단의 MPR 버튼 터치를 통하여 볼륨의 x축, y축, z축에 대한 MPR 영상을 출력 가능하다.



Figure 10. Rendering type converted by a user event
(Left top: MIP, Right top: MPR_X, Left bottom: MPR_Y, Right bottom : MPR_Z)

5. 성능 평가

본 연구에서 성능 평가에 사용한 볼륨 데이터는 CT에서 획득한 하반신(512×512×1165)과 머리(512×512×552) 그리고 복부(512×512×300)데이터를 사용하였다. 결과 생성 및 전송에 사용하는 출력 영상의 해상도는 512×512이다.

실험은 세 가지로 기획하였다. 첫 번째 실험은 전체적인 성능 분석으로 다수의 사용자가 접속했을 경우 각 사용자의 가시화 속도를 측정하였다. 실험 방법은 여러 사용자가 서버에 접속했을 때 지속적으로 서버에 이벤트를 전송하고 이에 대한 결과 영상을 가시화 하는데 걸리는 평균 시간을 측정하였다.

Table 1. Rendering speed at client side (fps)

Data type	1 People	3 People	5 People
Leg Data (512×512×1165)	56	48	37
Head Data (512×512×552)	60	53	42
Abdomen Data (512×512×300)	62	57	49

표 1은 볼륨 가시화 속도를 접속 사용자 수에 따라 비교한 표이다. 단일 사용자 환경의 경우 데이터의 크기에 관계 없이 비슷한 수준의 속도를 나타내고 있다. 그리고 동시 접속자가 증가하여도 데이터의 크기에 관계 없이 비슷한 추이를 나타낸다. 하지만 사용자가 증가함에 따라 각 사용자 별 가시화 속도가 감소하는 것을 볼 수 있다. 사용자가 증가 할수록 속도가 저하되는 이유는 서버가 한 명의 사용자에게 대한 가시화 연산을 수행하는 동안 OS스케줄링에 따라 나머지 사용자는 대기하기 때문이다. 하지만 사용자가 5명까지 증가하여도 30fps 이상의 속도를 보여 실시간 가시화에는 무리가 없는 정도이다.

두 번째 실험은 모바일 사용자가 느끼는 속도 향상을 검증하기 위해 제안 방법(터치 이벤트 전송 횟수 절감)과 기존 방법(제안 방법을 적용하지 않았을 경우)의 성능을 분석하였다. 실험 방법은 사용자의 터치 이벤트를 서버에 전송하고 두 장의 영상이 전송되기까지 시간을 측정하여 표 2에 제시하였다. 기존 방법은 두 번의 이벤트를 전송하였으며 제안 방법은 한 번의 이벤트만을 전송하면 된다. 그리고 기존 방법과 제안 방법의 평균 가시화 시간을 비교하여 표 3에 제시하였다.

Table 2. Latency time at client side (ms)

Data Type	Existing Method	Proposed Method
Odd number image	17	16
Even number image	18	5

Table 3. Rendering speed comparison (fps)

Data Type	Existing Method	Proposed Method
1 People	60	60
3 People	38	53
5 People	25	42

표 2는 사용자 응답속도에 대한 비교 표이다. 기존 방법은 각각의 영상을 전달받기까지의 평균 시간이 비슷하다. 제안 방법에서 짝수 번째로 영상을 생성할 때는 서버에서 메시지 처리의 대기시간이 제거되어, 매우 빠른 속도로 영상을 얻을 수 있다. 그 결과로 사용자의 가시화 속도가 향상된다.

표 3은 가시화 속도를 사용자 수에 따라 비교한 표이다. 제안 방법을 통해 가시화 속도가 대폭 향상 되었으며 사용자의 접속 수가 증가 할수록 가시화 속도가 향상된다. 표 2에서 응답시간이 평균 1.67배 (= 35 / 21) 향상되었는데, 그에 따라 5명

의 사용자가 서버를 사용할 경우 가시화 속도가 그와 유사한 1.68배(= 42 / 25) 향상된 것을 확인할 수 있다. 서버를 사용하는 클라이언트의 수가 적어 서버 성능에 여유가 있을 경우 가시화 속도는 화면 갱신 속도와 동일한 최대치인 60 fps가 측정되었으나, 클라이언트의 수가 증가하면서 서버의 처리량이 많아지게 되면 본 연구의 제안 방법인 메시지 전송 생략으로 가시화 속도를 크게 향상시킬 수 있다.

세 번째 실험은 사용자의 가시화 속도 향상을 검증하기 위해 인코딩 형식에 따른 성능을 분석하였다. 실험 방법은 서버에서 영상을 생성하고 원시 영상(raw image) 형태로 네트워크를 통해 전송했을 경우와 인코딩 작업을 수행했을 경우로 나누었다. 이때 사용자가 영상을 수신하는데 걸리는 소요시간을 측정 하였다.

Table 4. Encoding time and rendering time

Data Type	Encoding Time (ms)	Data Size (kbyte)	Rendering Time (ms)
RAW (512×512)	0	1048	573
JPEG (512×512)	2	7	12
PNG (512×512)	21	118.2	124

표 4를 보면 원시 자료의 경우 인코딩 시간이 소요되지 않지만 데이터의 크기가 매우 크다는 것을 알 수 있다. 이와 반대로 JPEG영상의 경우 인코딩 시간이 소요되는 반면 데이터의 용량이 작다. 가시화 시간의 경우 원시자료대비 JPEG 영상이 매우 빠른 것을 알 수 있다. 원시 자료의 가시화 시간이 느린 원인은 별도의 인코딩 시간은 소요되지 않지만 데이터의 용량이 크기 때문에 네트워크 대역폭에 의한 병목현상이라고 판단 된다. JPEG 인코딩 방법은 영상에 대한 압축을 수행하기 때문에 적은 양의 데이터를 전송하게 되어 상대적으로 가시화 속도가 빠르다. 이때 JPEG 인코딩에 소요되는 시간은 매우 적기 때문에 병목현상의 원인이 되지 않는다.

추가적으로 PNG 영상의 경우 인코딩 속도가 느리고 상대적으로 데이터의 크기가 JPEG에 비해 크다. 하지만 그림 11과 같이 JPEG영상 과 PNG 영상 간의 화질 차이가 크지 않은데다가, PNG 영상은 사용자의 이벤트가 종료되고 자세하게 관찰할 경우 단 한 번만 전송하기 때문에 이에 대한 속도 저하는 감내할 수 있는 정도이다.

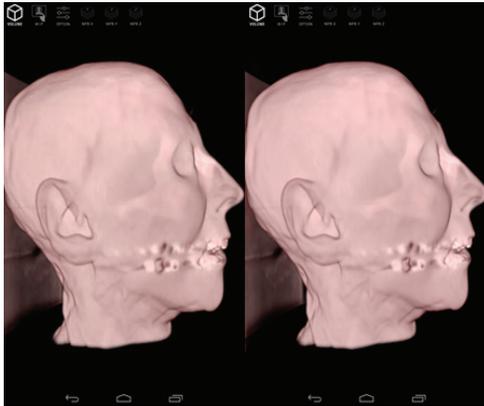


Figure 11. Encoding image(Image size 512 x 512)
Left : JPEG, Right : PNG

6. 결론

본 연구에서는 클라이언트 서버를 이용한 볼륨 렌더링 시스템의 설계 및 구현에 대해 설명하였다. 클라이언트는 대중적인 안드로이드 기반의 모바일 환경이며 서버는 GPU를 장착한 일반 PC로서 다중 사용자에게 대응 가능하다. 개발된 시스템은 사용자가 업로드한 볼륨 데이터의 미리 보기 및 선택, 볼륨 가시화, 볼륨 데이터의 기하변환, TF 변환, 렌더링 타입 선택 등과 같은 다양한 기능을 포함하고 있으며, 이를 편리하게 사용할 수 있도록 터치 이벤트를 활용한 GUI 환경을 개발했다. 렌더링 서버의 경우 가시화 영상의 품질을 향상 시키기 위해서 선-적분 볼륨 가시화 방법을 사용했으며, 볼륨 가시화 알고리즘 가속화를 위해 빈 공간 도약 기법 및 조기 광선 종료 기법을 사용하였다. 또한 모바일 디바이스의 가시화 속도를 향상 시키기 위해 자바 이외의 언어로 만들어진 모듈과 자바 간의 상호작용 할 수 있게 정의한 인터페이스인 JNI를 사용하여 네이티브 코드 환경에서 가시화를 수행 하였다.

그리고 사용자에게 대한 반응 속도 향상을 위해 본 연구는 두 가지 방법을 제안하였다. 우선 클라이언트는 사용자 이벤트를 일부만 서버에 전송하고 서버에서는 이를 보간 하여 결과 영상들을 클라이언트로 송신하도록 구현하였다. 그리고 실시간으로 영상을 전송하기 위해 네트워크 대역을 줄이기 위한 방법으로 영상 압축을 적용하였다. 사용자가 영상을 자주 요청하는 경우, 손실 압축인 JPEG영상을 전송하고, 사용자 이벤트가 종료되면 영상의 품질을 보장하기 위해 비 손실 압축인 PNG 영상을 사용자에게 전송한다.

본 연구를 통하여 사용자는 모바일 기기의 하드웨어 그리고 시간 및 장소에 구애 받지 않고 대용량 데이터의 실시간 가시화가 가능하였다.

감사의 글

This Research was financially supported by Hansung University.

Reference

- [1] M. Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, Vol. 8, pp. 29-37, 1988.
- [2] P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *ACM SIGGRAPH 1988 Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 51-58, 1988.
- [3] Noon C. J. "A Volume Rendering Engine for Desktops, Laptops, Mobile Devices and Immersive Virtual Reality Systems using GPU-Based Volume Raycasting" PhD thesis, 2012
- [4] D. Koller, M. Turitzin, M. Levoy, M. Marini, G.Croccia, P. Cignoni and R. Scopigno, "Protected interactive 3D graphics via remote rendering," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)*, Vol. 23, No. 3, pp. 695-703, 2004.
- [5] A. Fuhrmann, B. Ozer, L. Mroz and H. Hauser, "VR2 interactive volume rendering using PC-based virtual reality," *Tech Rap 2002-014. VRVis*, 2002.
- [6] L. Parsonson, L. Bai, S. Grimm, A. Bajwa, and L. Bourn. "Medical imaging in a cloud computing environment," In *International Conference on Cloud Computing and Services Science*. to appear, 2011.
- [7] Mobeen, M. M., and Feng, L., "Mobile visualization of biomedical volume datasets," *Journal of Internet Technology and Secured Transactions*, Vol. 1, Issues 1/2, 2012
- [8] Lamberti, F., Zunino, C., Sanna, A., Fiume, A., and Maniezzo, M. "An Accelerated Remote Graphics Architecture for PDAs," In *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 55-61, 2003.
- [9] Lamberti, Fabrizio, and Andrea Sanna. "A solution for displaying medical data models on mobile devices." *SEPADS 5*, 1-7, 2005
- [10] K. Enge, M. Hadwiger, J.M.Kniss, C.Rezk-Salama, and D.Weiskopf, *Real-Time Volume Graphics*, Wellesley, Massachusetts, 2006
- [11] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 2, pp. 99-108, Jun, 1995.
- [12] T. Kim, *OpenCL programming*, Hanbit, Seoul, Korea, 2012.
- [13] Y. Jung, *CUDA parallel programing*, Freelec, Seoul, Korea, 2011.
- [14] F. Hernell, P. Ljung, and A. Tnnerman, "Local Ambient Occlusion in Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, No. 4, pp. 548-559, 2010.
- [15] L. Marsalek, A. Hauber, and P. Slusalled, "High-Speed Volume

- Ray Casting With CUDA," *IEEE Symposium Interactive Ray Tracing 2008*, pp. 185, 2008.
- [16] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Vol. 9, No. 3, pp. 245-261, 1990.
- [17] W. Li, K. Mueller, and A. Kaufman, "Empty Space Skipping and Occlusion Clipping for Texture-Based Volume Rendering," *Proceedings of IEEE Visualization Conference (2003)*, pp. 317-324, 2003.
- [18] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 9-17, 2001.
- [19] BitmapFactory <http://developer.android.com/index.html> (accessed Mar, 26, 2015)
- [20] libjpeg-turbo <http://libjpeg-turbo.virtualgl.org/> (accessed Mar, 26, 2015)

〈저자 소개〉



이용규

- 2014년 한성대학교 정보시스템공학과 학사
- 2014년~현재 한성대학교 정보시스템공학과 석사 과정
- 관심분야: Volume Rendering, Medical Image Visualization, GPU Multi processing



계희원

- 1999년 서울대학교 전산학 학사
- 2001년 서울대학교 컴퓨터공학 석사
- 2005년 서울대학교 컴퓨터공학 박사
- 2007년~ 현재 한성대학교 정보시스템공학과 부교수
- 관심분야: 실시간 가시화, 의료영상처리