# Remote Volume Rendering Pipeline for mHealth Applications

Ievgeniia Gutenko*, Kaloian Petkov*, Charilaos Papadopoulos*, Xin Zhao*, Ji Hwan Park*, Arie Kaufman*, and Ronald Cha**
*Computer Science Department, Stony Brook University, Stony Brook, NY, USA, 11794
**Samsung Research Lab at Dallas, Richardson, TX, USA, 75082

## ABSTRACT

We introduce a novel remote volume rendering pipeline for medical visualization targeted for mHealth (mobile health) applications. The necessity of such a pipeline stems from the large size of the medical imaging data produced by current CT and MRI scanners with respect to the complexity of the volumetric rendering algorithms. For example, the resolution of typical CT Angiography (CTA) data easily reaches 512^3 voxels and can exceed 6 gigabytes in size by spanning over the time domain while capturing a beating heart. This explosion in data size makes data transfers to mobile devices challenging, and even when the transfer problem is resolved the rendering performance of the device still remains a bottleneck. To deal with this issue, we propose a thin-client architecture, where the entirety of the data resides on a remote server where the image is rendered and then streamed to the client mobile device. We utilize the display and interaction capabilities of the mobile device, while performing interactive volume rendering on a server capable of handling large datasets. Specifically, upon user interaction the volume is rendered on the server and encoded into an H.264 video stream. H.264 is ubiquitously hardware accelerated, resulting in faster compression and lower power requirements. The choice of low-latency CPU- and GPU-based encoders is particularly important in enabling the interactive nature of our system. We demonstrate a prototype of our framework using various medical datasets on commodity tablet devices.

**Keywords:** remote volume rendering, thin-client architecture, high quality streaming video, CPU/GPU encoders, medical applications, mHealth, mobile health.

## 1. DESCRIPTION OF PURPOSE

With widely available wireless network connectivity, the proliferation of mobile devices, and the constant increase in the graphics processing power, it is intuitive to remotely process large amounts of medical imaging data on the powerful server, while allowing fast and responsive streaming to mobile devices. The development of these components individually, enables the framework presented in this paper, which is significantly advanced when compared to previous attempts of remote rendering [1,2].

First, we consider the server side for the processing of medical data that is essentially the driving force in our system. The commodization of GPUs has resulted in the development of clustered visualization applications. While many of those applications involve off-line rendering, for example when dealing with very large scientific datasets or complex movie scenes, remote visualization has been used in certain interactive applications as well. One such example is cloud-based gaming services, where high-quality images are generated in a data center, streamed over a network connection, and the user controls the game on the client side.

Secondly, due to the significant increase in the role of mobile computing over the past few years, we are able to consider advanced mobile devices as suitable clients for our system architecture. Current mobile devices, including various smartphones and tablet devices, have powerful CPU and GPU characteristics that make them suitable platforms for various mHealth applications. Their configuration, computing capability, display quality and resolution are comparable to desktop counterparts available only a few years ago. However, due to the large amount of volumetric data, transferring the data and rendering it entirely on the device is not always possible [3,4]. The portability and always-on connectivity of these devices allows our system to stream images rendered on the server to the mobile device, and permits a medical doctor or health care professional to conduct the diagnostic process and follow up without being

constrained to a workstation computer in the hospital facility. Therefore, mobile devices are extremely promising as the client side for a remote volume rendering framework.

We leverage this technology to provide high-quality volume rendering targeted at the medical applications on low-power mobile devices. This framework has not been presented or published elsewhere. The main contributions of this work are:

- Fast remote volume rendering pipeline for mHealth applications capable of handling large volumetric datasets (up to $2048^3$);
- High quality responsive streaming synthesized to mobile clients, using hardware-accelerated encoding and decoding techniques;
- Evaluation on mobile health applications such as Computed Tomography (CT) colonography, exploration of other large CT datasets.

# 2. METHOD

Our remote volume rendering system is an extension of our existing distributed visualization software and offers a variety of configurations. In the case of remote volume rendering, the two principal components of our system are the image-generating *server* and a thin *client* that accepts and transmits user input and also streams images from the server. In this section, we outline the implementation details of each component.

## 2.1 Server Implementation Details

The server component implements a volume rendering pipeline that supports a variety of modalities (raycasting, texture slicing, 1D and 2D transfer functions, different compositing methods, etc). The server can operate in *local* mode, outputting the final image directly to the screen. For remote rendering, the server instead operates in *off-screen* mode. In this configuration, the server instead outputs the rendering result to a texture, via an OpenGL framebuffer object (FBO). Additionally, in this mode of operation, the server renders at a fixed internal resolution, rather than resizing the OpenGL viewport based on the size of the on-screen window. Also, the aspect ratio of the rendering is decoupled from the on-screen window and is instead transmitted to the server by the client application on startup and when changes in the client's window size occur. The remainder of the volume rendering pipeline is agnostic to the server's mode of operation.

In the final stage of the image synthesis kernel, the server performs a color-space conversion from the RGB output to YUV 4:2:0 on the GPU. YUV 4:2:0 is ubiquitously accepted as an input color space for a number of hardware-accelerated H.264 encoders. In our system's case, the NVENC (for modern Kepler GPUs) also supports the YUV 4:4:4 format that preservers the color at full resolution. Finally, the server reconverts the volume rendering result to RGB in a rapid, screen-space, pass and displays it locally for debug and visualization purposes.

The color-transformed framebuffer is passed on to the encoder subsystem for conversion into an H.264 video stream. This subsystem supports the NVCUVENC and NVENC [8] video encoders, which allow for performant coding of the images on the GPU. The video stream is exposed over the Real-Time Streaming Protocol (RTSP), implemented using the LIVE555 library [9]. A side-benefit of this approach is that the visualization session can be observed not only by the client application (outlined below), but also by almost any commodity device with media streaming support (such as an ARM-based smartphone). A disadvantage of RTSP-based streaming is the fact that it does not leverage the resiliency features of NVENC. We plan on developing a custom streaming protocol that takes advantage of these features, permitting clients to notify the server of transmission errors. In this situation, future frames would be encoded without

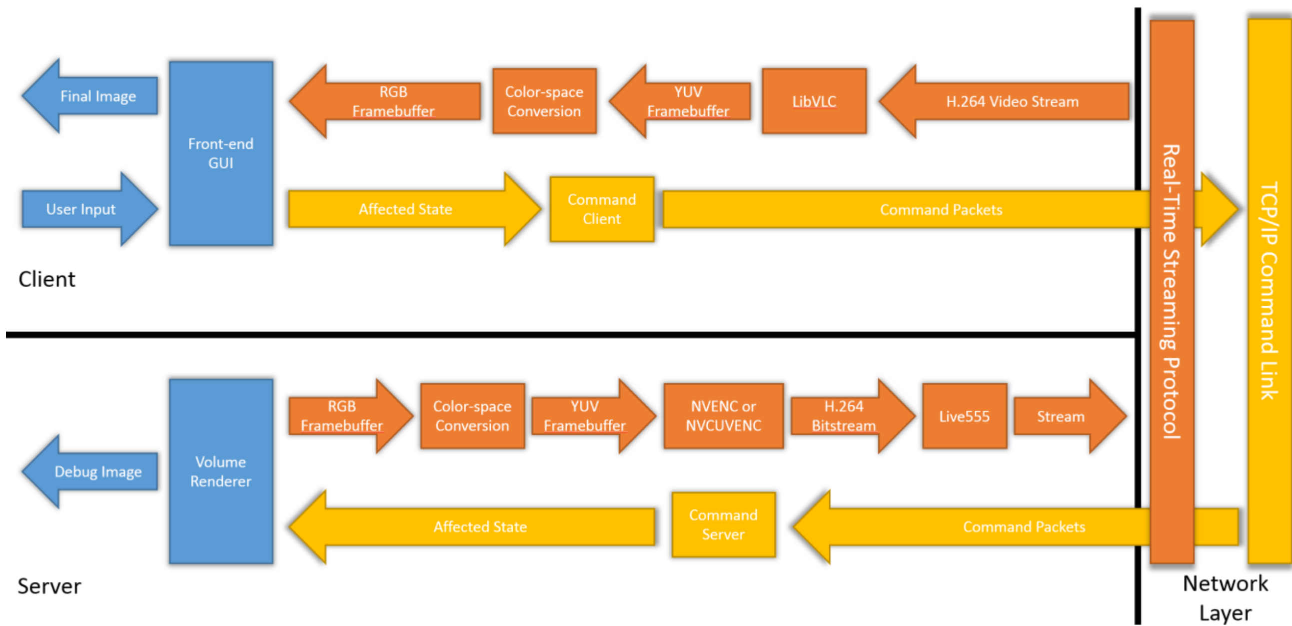



Figure 1: Block diagram outlining the connectivity and flow of data between components of our volume rendering pipeline. Blocks and arrows in orange relate to the streaming of volume renderings over an H.264 video stream, while the command infrastructure is represented in yellow. User-facing inputs and components are annotated in blue.

references to past corrupted frames. In our current implementation, RTSP provides resiliency to such network transmission errors at the expense of input latency.

Finally, the server application instantiates a TCP/IP server, listening for incoming connections on a predetermined port. Successful connection establishes a *command link* between the server and a single client instance, over which *command packets* are exchanged. Incoming packets are processed based on a header byte, with different headers denoting packets of different content. Our system decouples the transmission of different aspects of the render state in order to reduce network overhead. For example, manipulating the virtual camera results in only those parameters being transmitted to the server (rather than communicating the entire render state, including compositing settings, transfer functions, etc.). It is also worth noting that even a single camera manipulation can generate tens of command packets. Rather than repainting the scene once for every incoming command packet, the server maintains a steady refresh rate (30 or 60 frames per second in our implementation) and each frame is generated using the aggregate information from all command packets received up to that point in time.

### 2.2 Client Implementation Details

The client component of our system presents a graphical user interface to the user that is visually indistinguishable from a server instance running in local mode. All relevant volume rendering functionality is exposed via mouse, keyboard and multi-touch interaction modalities. The client maintains a local copy of the render state variables and transmits the appropriate command packets when parts of the state are affected over TCP/IP.

The key subsystem of the client is an H.264 network streaming client, implemented using LibVLC [10]. The utilization of this pervasive media playback framework allows for hardware acceleration of H.264 decoding on certain hardware platforms via the following APIs:

- Intel Media SDK [11] – interfaces with the Intel GPUs available in x86 tablets, such as the Microsoft Surface. The SDK provides hardware-accelerated H.264 decoding and integrates with OpenCL to provide additional processing of the video stream on the client. One major disadvantage is that this SDK is a hybrid system with major computation load on CPU.
- NVCUVID [8] – library for accessing the video decoding functionality of NVIDIA GPUs. Depending on the GPU capabilities, the decoding may be implemented in the CUDA language, or using power-efficient on-board hardware. The resulting images can be further processed in CUDA or combined with client-based rendering in OpenGL.

The decoded framebuffer is presented to the application in YUV format, which is color-converted to RGB on the GPU via an OpenGL 2.0 pixel shader and then presented to the user. This color-space conversion is the most advanced GPU feature required by the client application and it is supported on all x86-compatible integrated GPUs that have been released in recent years. Consequently, the client application can be executed on a vast range of devices and poses minimal hardware requirements. A block diagram illustrating the architecture of our mobile volume rendering pipeline can be seen in Figure 1.

## 3. RESULTS

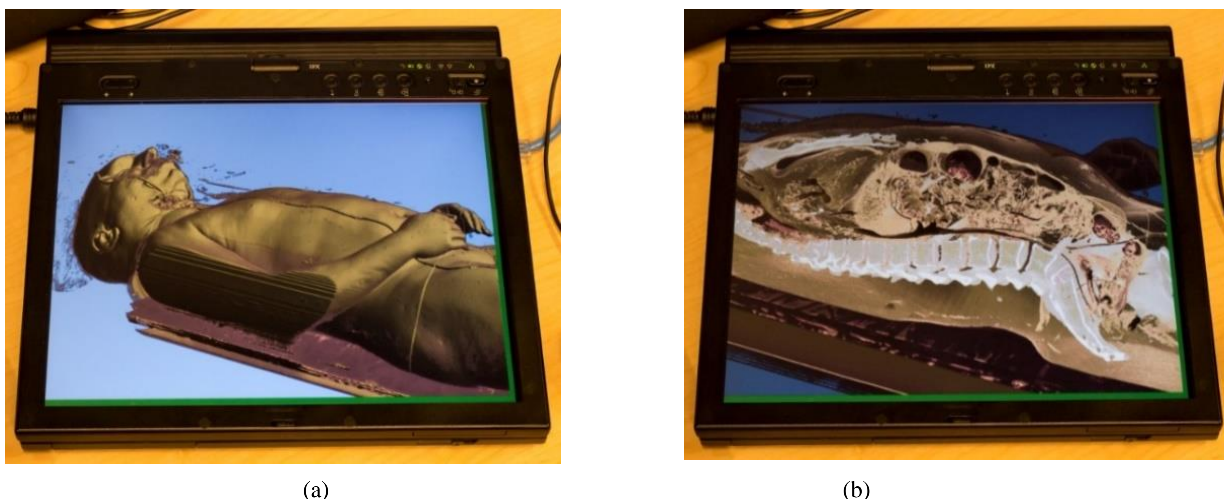
Our volume rendering system supports a variety of visualization techniques, which are also available during video streaming. We focus our evaluation specifically on features that cannot be implemented on low-powered devices as follows:

- Smoother rendering during interaction – Mobile devices often struggle with rendering volumes at high screen resolutions. The visuals of some immersive interactive applications, such as Virtual Colonoscopy, cover the entire viewport the majority of time, making local rendering option unattractive. Contrarily, our client-server approach is not constrained in this way.
- Higher-order reconstruction filter – we use a cubic filter that is 3 times more expensive than the standard trilinear filter, but provides a higher image quality. The usage of such a filter would be prohibitive when rendering locally on a mobile device.
- Large datasets – our system handles volumes up to 2048^3 at interactive rendering speeds. Rendering such volumes locally on a mobile device would be very difficult, due to space and bandwidth constraints.

(a)

(b)

Figure 2: Remote visualization of (a) CT colonography and (b) CT lung volume data.



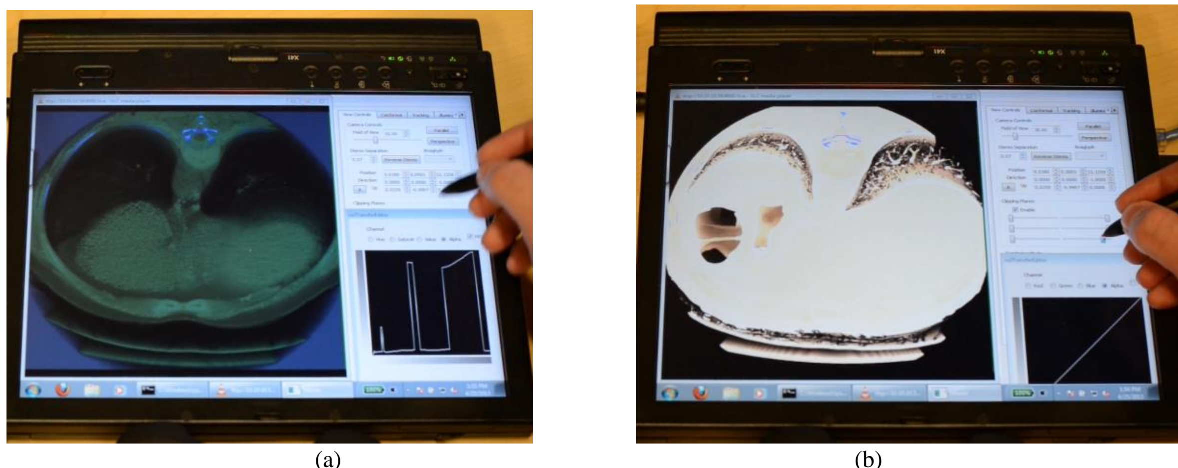
(a) (b)

Figure 3: Remote visualization of the Visible Human dataset with (a) a full body view and (b) a sagittal view.

Our evaluation is performed on the following hardware: cluster of 6 Dell Precision T7600 workstations with dual 6-core CPUs, 64GB of memory and NVIDIA Quadro K5000 GPU as server; and IBM Thinkpad X41 tablet as client. The tablet is significantly slower than modern devices and does not offer any hardware video decoding support. However, it was chosen purposely to show that our framework is not constrained to the latest mobile devices as clients.

We evaluate the results based on several potential applications. The prototype of a CT Colonography (Virtual Colonoscopy) applications (such as [5,6]) is evaluated and shown in Figure 2(a). The volume rendering is performed on the server side, encoded in the H.264 format and streamed to the client side tablet device. The interactive exploration of the lung data is shown on the Figure 2(b). The data resolution is 512x512x431 which still allows us to provide a 60 fps video stream even with the high order reconstruction filter. The transfer function is optimized for visualization of the colon wall, yet it also segments the lung structures.

Figure 3 shows snapshots of the interactive exploration of the Visible Human [7], which is a 512x512x1877 volumetric dataset. The NVIDIA Quadro K5000 in the server side can provide smooth interaction for the visualization under a variety of rendering modes (isosurface rendering of the skin and visualization of internal tissues). Figure 4 illustrates interaction of the user with a client device allowing her to change parameters of the visualization.



(a) (b)

Figure 4: Remote visualization of the cardiac dataset. (a) 1D transfer function editing on the client. (b) Clipping plane positioning on the client.

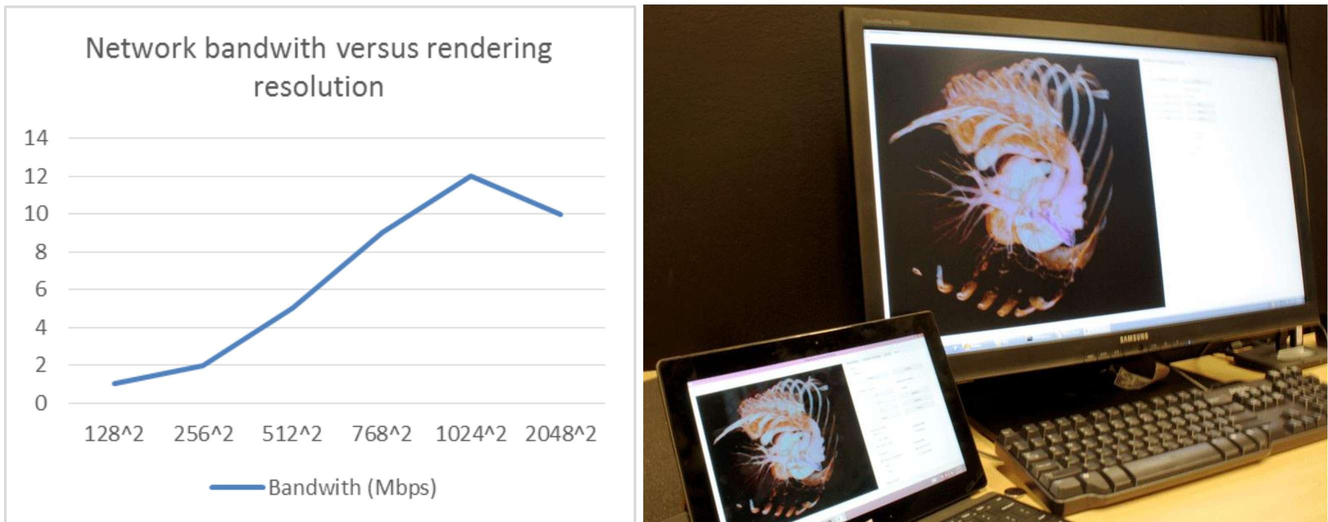


Figure 5: Left: Chart summarizing our network bandwidth utilization benchmark. Right: View of our test system, showing a Microsoft Surface tablet device and a high-end workstation, running the client and server components of our system with synchronized views.

Additionally, we measured the network bandwidth required for transmitting the H.264 stream, as a factor of the internal rendering resolution of the server. For this benchmark, we utilized a higher-end tablet device (Microsoft Surface) that would benefit from the additional detail of increased rendering resolutions, due to its 1080p display. We ensured that the synthesized image covered the majority of the view frustum and applied a colorful transfer function to the volume (in order to introduce some entropy to the chroma channels of the framebuffer). We observed a mostly linear increase in bandwidth utilization. For an optimal rendering resolution of 1024^2, we recorded an average bandwidth utilization of approximately 12 megabits / second, while rendering at 30fps. It is worth noting that for higher rendering resolutions (such as 2048^2), network utilization actually drops, as the server is not able to generate frames at a high-enough framerate for particular rendering settings. Our measurements, along with an illustration of the benchmark setup are visible in Figure 5.

## 4. CONCLUSIONS

In this paper, we demonstrate a remote volume rendering system that leverages high-end workstations to generate the images based on user input and to also encode and stream the results in the H.264 video format to a mobile device. The volume rendering image synthesis happens on the server side, on a dedicated workstation or GPU cluster. Our system can generate multiple views of the volume data, and supports multiple video streams with different bitrates and resolutions for different client devices. Visualization clients can interface with a session either via a commodity media streamer (without support for user interaction) or through a thin-client application that exposes the full user interface and transmits inputs via TCP/IP. Our prototype system can handle various large and complex volume data and provide high quality volume rendering at real-time speeds.

Looking forward, we are expanding our remote rendering framework in two ways. First of all, we are exploring novel techniques for natural interaction with the volumetric visualization (such as augmented reality and various forms of gestural interaction). Additionally, we are investigating various avenues for further increasing the rendering performance on the client side and reducing the network overhead associated with H.264 streaming. For example, our current pipeline performs the volume rendering at a fixed internal resolution, which remains constant throughout the session. We are investigating adaptive rendering schemes that dynamically adjust the volume rendering resolution based on available bandwidth and the complexity of the image.

## ACKNOWLEDGEMENTS

This work has been supported by grants from NSF IIS0916235, IIP1069147 and CNS0959979, Samsung Research Labs, and the Center of Excellence in Wireless and Information Technology (CEWIT) at Stony Brook University. The Visible Human dataset is courtesy of NLM and the cardiac dataset is courtesy of Stony Brook University Medical Center.

## REFERENCES


[1] S.J. Jeong and A. Kaufman. "Interactive wireless virtual colonoscopy," The Visual Computer, 23(8), 545-557 (2007).

[2] L. Campoalegre, P. Brunet, and I. Navazo. "Interactive Visualization of Medical Volume Models in Mobile Devices," Personal and Ubiquitous Computing, 17(7), 103-1514 (2013).

[3] J. Noguera, J. Jimnez, C. Ogyar, and R. Segura. "Volume Rendering Strategies on Mobile Devices," In International Conference on Computer Graphics Theory and Applications, 447-452 (2012).

[4] J. Noguera, and J. Jiménez. "Visualization of Very Large 3D Volumes on Mobile Devices and WebGL," In 20th WSCG International Conference on Computer Graphics, Visualization and Computer Vision, 105-112 (2012).

[5] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. "Virtual Voyage: Interactive Navigation in the Human Colon," In Proceedings of the 24th annual conference on Computer Graphics and Interactive Techniques, SIGGRAPH, 27-34 (1997).

[6] H.M. Fenlon, D.P. Nunes, P.C. Schroy, M.A. Barish, P.D. Clarke, and J.T. Ferrucci. "A Comparison of Virtual and Conventional Colonoscopy for the Detection of Colorectal Polyps," New England Journal of Medicine, 341(20), 1496-1503 (1999).

[7] Spitzer, Victor, Michael J. Ackerman, Ann L. Scherzinger, and David Whitlock. "The Visible Human Male: a Technical Report," Journal of the American Medical Informatics Association, 3(2), 118-130 (1996).

[8] NVCUVENC and NVENC video encoders, NVUVID. https://developer.nvidia.com/nvidia-video-codec-sdk (April 2013)

[9] LIVE555 library, http://www.live555.com/liveMedia/ (April 2013)

[10] LibVLC library, http://libvlcnet.sourceforge.net/ (April 2013)

[11] Intel Media SDK, http:// software.intel.com/en-us/vcsource/tools/media-sdk (April 2013)