

# Server-Aided 3D DICOM Image Stack Viewer for Android Devices

Menghe Zhang, Jürgen P. Schulze; University of California San Diego, La Jolla, CA, US

## Abstract

*Digital Imaging and Communications in Medicine (DICOM) is an international standard to transfer, store, retrieve, print, process and display medical imaging information. It provides a standardized method to store medical images from many types of imaging devices. Typically, CT and MRI scans, which are composed of 2D slice images in DICOM format, can be inspected and analyzed with DICOM-compatible imaging software. Additionally, the DICOM format provides important information to assemble cross-sections into 3D volumetric datasets. Not many DICOM viewers are available for mobile platforms (smartphones and tablets), and most of them are 2D-based with limited functionality and user interaction. This paper reports on our efforts to design and implement a volumetric 3D DICOM viewer for mobile devices with real-time rendering, interaction, a full transfer function editor and server access capabilities. 3D DICOM image sets, either loaded from the device or downloaded from a remote server, can be rendered at up to 60 fps on Android devices. By connecting to our server, users can a) get pre-computed image quality metrics and organ segmentation results, and b) share their experience and synchronize views with other users on different platforms.*

## Introduction

Magnetic resonance (MR) and computed tomography (CT) imaging devices have been incorporated into mainstream radiology practice and are performed daily for a wide range of clinical indications. DICOM datasets are widely used in the medical field. Traditionally, a radiologist reviews a patient's imaging data one 2D image at a time and selects those that show the best views of the disease among hundreds of images, using them as the basis for a medical report. More recently, 3D volumetric visualization offers a more comprehensive view of the medical imaging data. However, volumetric visualization is more compute-intensive than 2D imaging, thus viewing systems must process data particularly efficiently, and due to the complex nature of volumetric data sets they must be user-friendly to be useful.

There are many applications on desktop platforms for different usage scenarios, some of which use novel methods that take the 3D information into account that is stored in the image sets. We decided to develop our software for mobile devices to balance convenience and performance. Recent advancements in mobile chipsets enable us to do complex rendering in real-time. Our software makes 3D volume visualization more approachable for both specialists and non-specialists and the mobile nature of the devices makes it easier to communicate, for instance for doctors with their patients.

One problem we addressed is how to improve the 3D visualization of medical data by utilizing the available meta-data from the DICOM files. Another challenge was how to properly design the

user interface to provide the necessary visualization features without overwhelming the user with too much complexity.

We developed an Android application to load, view and interact with 3D medical volumes. We designed the core software modules to make it easy to support other hardware platforms. Some mobile DICOM Viewers provide access to hospital image servers (PACS) to retrieve datasets. We created a server running on a desktop, provides another option that lies in-between the mobile clients and remote PACS system so that we can store pre-processed MRI datasets and metadata, in order to minimize the computational load on the mobile device. Compared to previously existing DICOM viewers, the novel contributions of our work are:

- **Real-time experience:**Real-time volume rendering with a 2-stage process on mobile Android platforms.
- **Rich functionalities:**A client-server architecture which provides additional information including organ masks, colon center lines and image quality metrics. Those data allowed us to add features that are not found in other mobile medical data viewers.
- **Collaborative views:**Our server also provides the ability for users on different devices with different platforms (e.g., PC) can view the same data in the same exact way, to allow for cross-platform collaboration.

Potential use case scenarios for our applications are varied, for example, medical education, communication, surgical planning, and early disease detection.

## Related Work

### Volume Rendering

Volume rendering techniques, including direct and indirect rendering, have been developed for medical image visualization to create customizable images [19, 27]. Direct Volume Rendering (DVR) [16, 8] generates images of 3D volumes by using an optical model to map data values to optical properties, such as color and opacity [17]. DVR includes backward and forward methods. Ray-casting is an example of backward methods, which is performed pixel-by-pixel. This method was introduced by Kajiya and Herzen [14] and was later translated to the GPU [15].

In the medical field, this rendering technique is used to transform sets of 2D images into 3D volumes. Some previous work optimizes the three steps for the unique needs of clinical use and sometimes aim to help anticipate potential complications during surgery operations [7, 13]. For example, Fishman et al. [9] compared the popular methods of alpha blending and maximum intensity projection for the evaluation of CT angiographic data sets. These two techniques have different advantages and disadvantages when used in clinical practice, and it is important that radiologists understand when and how each method should be used.

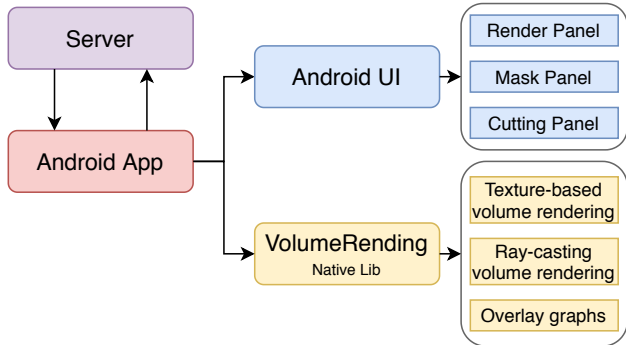


Figure 1: Software structure

## DICOM Viewers

DICOM viewers are often developed with a focus on one or more of the following functions: 1) quick viewing of medical images; 2) browsing the images from PACS servers. Some offer 3D rendering features as an option. According to a survey of DICOM viewers in 2016 [10], products that provide great 3D imaging with a high reputation are ImageJ [24], MicroView [12], MIPAV [25], and OsiriX Lite [23]. The recently released Horos for Mac is a free version of OsiriX, which is often considered as the best DICOM viewer for Macs. Other software applications, like Post-DICOM [21], Weasis [22], Ginkgo CADx, and Pro Surgical 3D allow high-quality rendering with high performance on PCs. In recent years, the web paradigm has introduced new visualization, sharing and interaction opportunities for medical images. Some web applications use WebGL to provide easy access to a quick 2D or 3D visualization of datasets [2, 1, 5], but 3D rendering performance cannot compete with native applications. In the mobile field, choices are more limited. Products on mobile devices like mRay [18], Symmetry [11], iPaxera [6] and simpleDICOM [3] provide a 2D view for each slice. Some of them enable the connection to a PACS server but do not have ways to store and share datasets. DroidRender [26] is considered one of the best options on Android that provides a fully functional 2D and 3D render engine, tissue segmentation and slicing.

## System Overview

Our software system consists of an Android app and an optional image server. Figure 1 shows an overview of our application. It is targeted towards Android, using the Java Native Interface to implement all the core functionalities in C++ for greater speed. For development and testing, we used a Samsung Galaxy S10 smartphone with Qualcomm’s Snapdragon 855 SoC and Adreno 640 GPU, API level 28. More powerful graphics chipsets will increase the rendering rate, produce more fluid screen updates, and overall better user experience.

## Volume Rendering and Display

We created a two-step rendering algorithm with a compute shader (Fig. 2). A 3D texture is prepared in stage one and rendered in the next stage. In stage one, the 3D texture is prepared with color and opacity mapping (transfer function), which are done in a compute shader. In stage two, this 3D volume is rendered with one of two methods: texture-based or ray-casting. This design separates the rendering parameters into two groups.

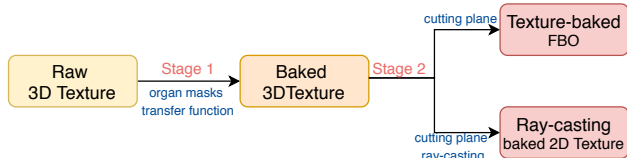


Figure 2: Illustration of our 2-stage rendering scheme

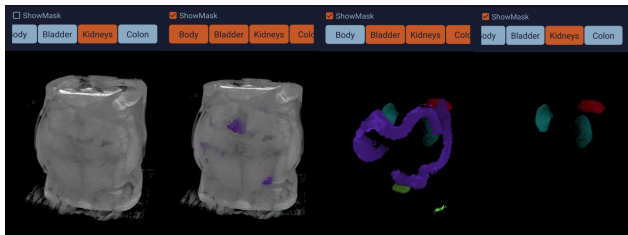


Figure 3: An abdominal MRI with organ segmentation. From left to right: (a) no masks; (b) body and organs with masks; (c) organs only; (d) kidneys and spleen only

## Stage One: Preparing the 3D Texture

For each volume data set, a 3-dimensional texture is created with data that is loaded gradually. Initially, users select a volume from the local or remote data set list, and the corresponding volume information is sent to the native application to allocate a buffer and prepare to set up 3D texturing. Once data loading finishes, the app creates a 3D texture and enters the second stage. At this point, additional data can still be assembled into the same buffer and sent to the GPU after all processes finish. In the compute shader, the following parameters are considered:

**Organ Mask:** Raw data in the texture represents 12 bits intensity values while each organ mask takes 1 bit for each unit of the texture (texel). So we create the texture in r32ui (red channel-32 bits-unsigned int type) format. The upper 16 bits are allocated for masks, and the lower 16 bits are for volume data. The system gets mask data asynchronously from the server and combines this part of the data with the intensity data and sends it to the GPU.

We later assign each organ a unique color. Figure 3 shows an abdominal MRI segmented into bladder, kidneys, colon, spleen, etc.

**Transfer Function:** We drew inspiration from the work of Chourasia et al. [4] to create effective intensity transfer functions for high dynamic range scalar volume data. Compared to their method, our work provides a 2-step approach: 1) a uniform enhancement of contrast and brightness; 2) simple primitives as widgets that allow users to design their own data opacity patterns. First, given a pair of contrast limits ( $C_{bottom}, C_{top}$ ) and the brightness  $C_b$ , the value of each voxel  $C_x$  is enhanced as:

$$C_{t1} = (C_x - C_{bottom}) / (C_{top} - C_{bottom}) + C_b, \quad (1)$$

$$(C_{bottom} \leq C_x \leq C_{top})$$

Secondly, multiple opacity widgets can be applied to pull out specific parts of the volume. For each widget, there are 5 parameters ( $v_o, v_l, vw_b, vw_t, v_c$ ) that determine the 6 control points (**lb, lm, lt, rb, rm, rt**) of the widget:

- Overall ( $v_o$ ): the highest value of the opacity ( $0 \leq v_o \leq 1.0$ ).

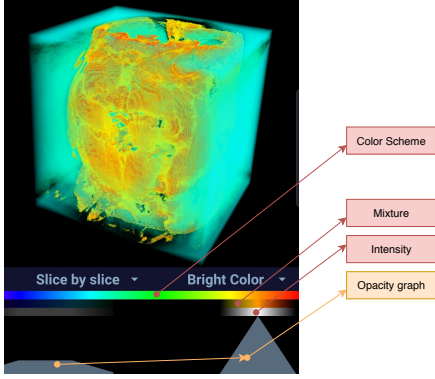


Figure 4: An abdominal MRI with opacity widgets and the color transfer function graph

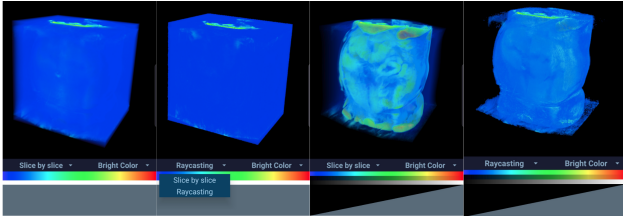


Figure 5: Volume rendering with texture-based and ray-casting methods. From left to right: (a) texture-based, opacity fully mapped; (b) ray-casting, opacity fully mapped; (c) texture-based, opacity linearly mapped; (d) ray-casting, opacity linearly mapped.

- Lowest Bound ( $v_l$ ): the lowest value ( $.0 \leq v_l \leq v_o$ ).
- Top Width ( $vw_t$ ): the top range value ( $.0 \leq vw_t \leq 2.0$ ).
- Bottom Width ( $vw_b$ ): the bottom range value ( $vw_t \leq vw_b \leq 2.0$ ).
- Center ( $v_c$ ): position of the highest value ( $.0 \leq v_c \leq 1.0$ ).

Then the 6 control points can be calculated accordingly. When multiple widgets add up, with the first transfer function being  $C_{t1}$ , we get

$$C_{i2} = \max(\text{Widget}_i(C_{t1})), \text{ where } i = 1, 2, \dots, n \quad (2)$$

Finally, we provide another two color transfer functions that map grayscale intensity values to colored values.

Figure 4 demonstrates the visualization of a volume with opacity widget graphs and a color transfer function bar. Users can get an intuitive view of how the parameters affect the transfer functions and how they affect the volume visualization.

### Stage Two: Render the Volume

We use Direct Volume Rendering (DVR) to show the entire 3D data volume and provide the options of texture-based (forward) and ray-casting (backward) volume rendering (Fig. 5).

**The Texture-based Method** projects each slice onto the viewing plane and blends according to capacities. All texture-mapped slices are stored in a stack and are blended back-to-front onto the image plane, which results in a semitransparent view of the volume. We then align the slices in object space parallel to the Z-axis, representing the nature of a stack of 2D images (Fig. 5ac). Therefore, viewers would look on the slice edges by rotating the

volume and see through the stacked slices. By projecting the voxels of the slice plane on pixels of the viewing plane, a pixel can lie in-between voxels, so its value is from bilinear interpolation.

**The Ray-casting Method** casts a primary ray from the camera into the volume, sampling the volume in regular intervals (Fig.5bd). The final color and opacity of the pixel are accumulated along the ray [20] by:

$$C_{out} = C_{in}(1 - \alpha(x_i)) + C(x_i)\alpha(x_i) \quad (3)$$

We also propose an adaptive sampling strategy, which takes more sampling steps in denser parts of the volume to enhance details. Algorithm 1 shows the pseudo-code of our adaptive sampling method. The goal is to perform the ray-casting from  $t$  to  $t + s$ . When sampled  $alphachannel > 0.01$ , we evaluate the alpha contribution from the current step: 1) If the contribution exceeds the threshold, we halve the step size and double the threshold and try to advance from  $t$  to  $t + \frac{s}{2}$ ; 2) If the contribution is below the threshold, we accumulate color and alpha and adjust step size and threshold based on the test of the last step. Figure 6 illustrates how this adaptive sampling during data accumulation is performed from  $t$  to  $t + s$

```

input :  $r_o, r_d, head, tail, s_u, thresh\_termine,$ 
          $thresh\_ignore, thresh\_init$ 
output:  $color(r, g, b, a)$ 

 $color \leftarrow 0;$ 
 $threshold \leftarrow thresh\_init;$ 
for  $t \leftarrow head$  to  $tail$  do
    if  $color.a \geq thresh\_termine$  then
        | terminate;
         $p \leftarrow r_o + r_d \times t;$ 
         $samplerd \leftarrow Sampling(p);$ 
        if  $samplerd.a > thresh\_ignore$  then
             $contrib \leftarrow (1.0 - color.a) * samplerd.a;$ 
            if  $contrib > threshold$  then
                |  $s \leftarrow s/2;$ 
                |  $threshold \leftarrow \min(2 \times threshold, 1.0);$ 
                |  $last\_succeed = false;$ 
            else
                |  $color.rgb \leftarrow color.rgb + (1 - color.a) \times$ 
                |  $samplerd.a \times samplerd.rgb;$ 
                |  $color.a \leftarrow contrib;$ 
                |  $t \leftarrow t + s;$ 
                | if  $last\_succeed$  then
                    | |  $s \leftarrow 2 \times s; threshold \leftarrow threshold/2$ 
                | else
                    | |  $last\_succeed = true$ 
            else
                |  $t = t + 4 \times s_u$ 

```

Algorithm 1: Accumulated ray-casting volume rendering with adaptive sampling step

### Display Auxiliary Information

Besides the original data volume, our system provides auxiliary information that helps to improve the user's ability to get the best view of the data.

**Cutting Plane:** We implemented a GPU-based cutting plane to

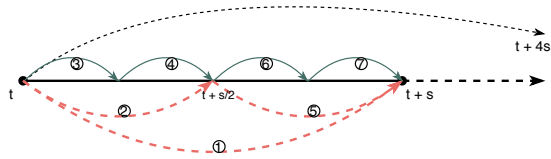


Figure 6: Illustration of adaptive sample steps

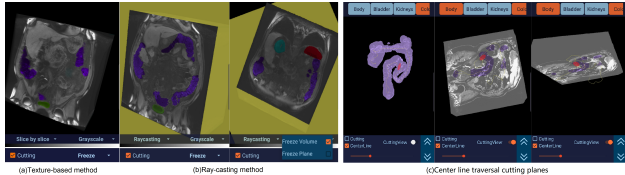


Figure 7: Cutting Plane in: (a) texture-based, (b) ray-casting volume rendering methods, and (c) in direction of center line

view details of a selected position with user interaction.

In texture-based rendering mode, a volume is composited by stacking a set of slices together so that users can view the data volume slice by slice as they do traditionally. While in ray-casting rendering mode, users have more flexibility to uncover occluded details in any direction and any depth.

Figure 7ab shows screenshots of a cutting plane with texture-based and ray-casting methods. Notice that in the ray-casting method, one can choose to keep the volume or cutting plane fixed in place and manipulate the other.

**Organ Masks:** Users can view the organs in volumetric form, 3D mesh form or both with organ masks generated by the server. We currently support up to seven different types of organs for different datasets. Figure 8 shows a data set with a colon mask in different visualization styles. With organ segmentation data from the server, our application can generate a surface mesh with the Marching Cubes algorithm in real-time via a compute shader. This step is normally done once at the beginning of the data loading process, unless the user decides to modify the data, in which case the organ surface representations have to be recreated. We down-sample the 3D dataset for faster generation of the organ outlines. The generated organ meshes can be displayed as solid meshes or wire-frame meshes. By displaying as wire-frames, users can see the organ as a volume plus its wire-frame surface structure. They can even see the center line inside of the organ, if it is available (currently only for colon and terminal ileum).

**Organ Center Lines:** The server generates center lines of colon and ileum for selected datasets (Fig. 8d). We also extended the cutting plane to allow for traveling continuously along the center line (Fig. 7c). By switching to the *cutting view*, the view direction will follow the center line to see through the cutting plane. In

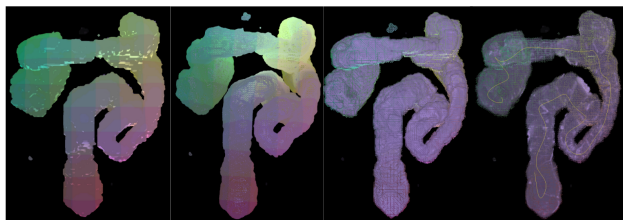


Figure 8: Colon shown as a) solid mesh; b) wire-frame mesh; c) volume with wire-frame mesh; d) volume with wire-frame and center line.

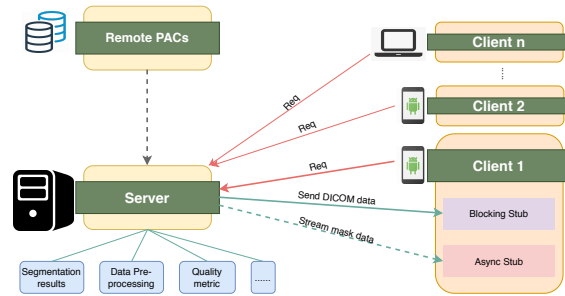


Figure 9: Server work flow

any view mode, users can always perform touch gestures to scale, rotate and move the data volume.

## gRPC Server

The gRPC server we created runs on a desktop and provides another option that lies in-between the mobile clients and the remote image server (PACS system) that 1) stores and pre-processes MRI datasets to generate auxiliary information and 2) provides communication between multiple clients or between the server and one of the clients.

With gRPC, a client application can directly call a method on a server as if it were a local object. This process can be called synchronously and asynchronously with different types of stubs. According to our need to browse and request datasets, we created the workflow as shown in figure 9.

## Pre-Processing and Data Transmission

As an optional data transfer station, when new datasets come from remote PACS system, the server will do off-line pre-processing on demand to:

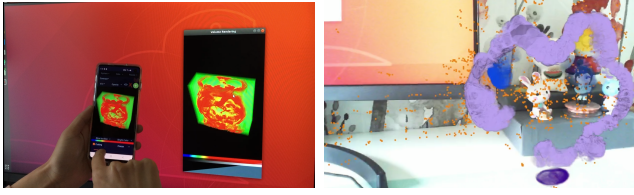
- **Build index files:** It keeps a dataset-index file to track the latest dataset and volume-index files to maintain the pre-processed information for each volume.
- **Generate quality metrics:** It quantifies the quality of the original DICOM images from the CT or MRI scanners and helps filter out low quality datasets. When users request to browse the available datasets, those quality metrics would help to selected the desired ones.
- **Generate organ masks and center lines**

## Share Views Across Platforms

Users can share views with other users on different platforms via the server (Fig. 10). To initiate this, one user's Android phone becomes the "host client" and controls the view; then other clients, either with mobile devices or desktop computers, connect to the server to get their views synchronized to the "host client". This functionality also works on AR devices, however, compared to the non-AR cases which share the whole view with others, it doesn't share the camera view but only the 3D dataset manipulations. This is because, in an AR environment, we want to give each user their own perspective on the dataset, so that a doctor can discuss a dataset with a patient.

## User Interfaces

Figure 11 demonstrates the design of our user interface: starting from the main menu, users can load data sets; when they



(a) Share view in non-AR context. (b) Share view in AR context  
Figure 10: Share view between Android phone and desktop.

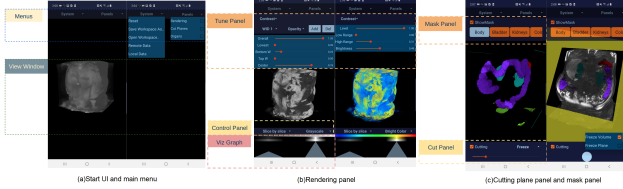


Figure 11: Demonstration of user interface design

look at details of a volume, they can always show or hide render panel, cutting plane panel and mask panel in any combination.

## Discussion

Our system provides two volume rendering methods to view the same datasets. **Qualitative results:** The texture-based method is fast for moderately sized datasets, and surface-based and volumetric representations can easily be combined. It works better when multiple opacity widgets are applied together. However, there are artifacts when a stack is viewed from close to 45 degrees (Fig. 5a). In comparison, the ray-casting method provides more details. Figures 12 and 13 compare views rendered with texture-based, ray-casting, and ray-casting with adaptive sampling methods. In most cases, adaptive sampling methods take more sample steps in a denser area, which enhances the details. However, if the selected area is non-convex, the ray-casting method eliminates the details inside the concave regions. With adaptive sampling steps, this elimination can get worse.

**Quantitative results:** In a real practice, users may use different combinations of rendering parameters for different visualization tasks, which highly affects the performance. We use a MRI volume dataset of size  $512 \times 512 \times 144$  and setup this application in a default contrast(0,1,1) and opacity(1,1,2,2,1) settings. Figure 14 compares the frames per second(FPS) of our adaptive raycasting method with different sampling step sizes. We observed that if the step size exceeds 0.025, there will be discernible artifacts on the volume.

Table 1 compares FPS of the application with different rendering methods. For this comparison, we set the sampling step

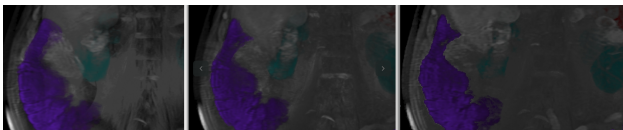


Figure 12: Compare volume rendered with texture-based, ray-casting, and ray-casting with adaptive sampling methods. From left to right: (a) texture-based; (b) ray-casting; (c) ray-casting with adaptive sampling

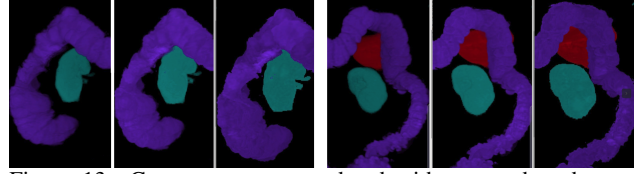


Figure 13: Compare organs rendered with texture-based, ray-casting, and ray-casting with adaptive sampling methods. Each from left to right: (a) texture-based; (b) ray-casting; (c) ray-casting with adaptive sampling

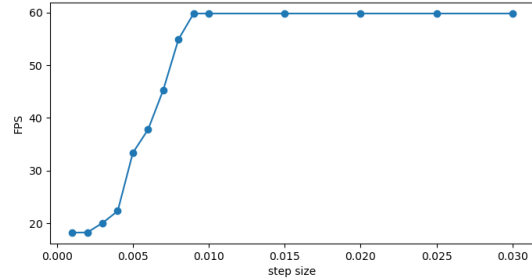


Figure 14: Rendering performance of adaptive raycasting method with different sampling step sizes

of raycasting rendering to 0.05. From the table we see that all the rendering methods could render the whole volume in nearly 60FPS. To render a region that contains more details, for example, when we render the masked volume which only shows the organs, raycasting method is much slower. For this dataset, the maximum number of triangles to render the organ mesh is  $512 \times 512 \times 144 \times 5 = 188743680$  and the application is able to seamlessly render the meshes while maintaining a near 60FPS.

## Conclusion

In this project, we created a server-aided DICOM viewer for Android mobile devices. Our results show that we can do real-time volume rendering with both texture-based and ray-casting methods to show 3D volumetric DICOM datasets. Users have flexibility to adjust the rendering parameters, cut into the volume, separate out major organs, and save or load those settings to or from a file. By connecting to our server, a user can also share the viewing parameters with others in real-time.

Future research should be devoted to the development of a better user interaction experience in augmented reality (AR) scenarios adapted to different AR devices with support for multi-user cooperation. Some of the functionalities of our server can still be optimized, such as to provide segmentation results on the fly with the request for specific parts of the tissue.

## Acknowledgments

This research was funded in part by a grant from The Leona M. and Harry B. Helmsley Charitable Trust. The authors would also like to thank the volunteers who provided their medical data

Table 1: Compare FPS among rendering methods

	volume	masked volume	masked mesh	masked volume+mesh
Texture-based	59.9	59.8	59.8	48.5
Raycasting	59.8	34.2	59.8	26.1
Raycasting(adp)	59.8	33.4	59.8	23.8

for us to work with under UCSD IRB protocol #182146.

## References

- [1] Felix-Constantin Adochiei, Radu-Ion Ciucu, Ioana-Raluca Adochiei, Sorin Dan Grigorescu, George Călin Seritan, and Miron Casian. 2019. A WEB Platform for Rendering and Viewing MRI Volumes using Real-Time Raytracing Principles. In *2019 11th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*. IEEE, 1–4.
- [2] Ander Arbelaz, Aitor Moreno, Luis Kabongo, Helen V Diez, and Alejandro García Alonso. 2017. Interactive Visualization of DICOM Volumetric Datasets in the Web-Providing VR Experiences within the Web Browser. In *International Conference on Information Visualization Theory and Applications*, Vol. 4. SCITEPRESS, 108–115.
- [3] Barton F Branstetter IV, Steven D Uttecht, David M Lionetti, and Paul J Chang. 2007. SimpleDICOM suite: personal productivity tools for managing DICOM objects. *Radiographics* 27, 5 (2007), 1523–1530.
- [4] Amit Chourasia and Jürgen P Schulze. 2007. Data centric transfer functions for high dynamic range volume data. (2007).
- [5] R Ciucu, F Adochiei, I Adochiei, F Argatu, CM Larco, and L Grigorie. 2019. A Real-Time WebGL Rendering Pipeline for MRI Using RayCasting Transfer Functions. In *International Conference on Nanotechnologies and Biomedical Engineering*. Springer, 529–534.
- [6] PaxeraHealth Corp. 2020. iPaxera. (2020). <https://apps.apple.com/us/app/ipaxera/id432861550>.
- [7] Bruno Fernandes de Oliveira Santos, Marcos Devanir Silva da Costa, Ricardo Silva Centeno, Sergio Cavalheiro, Manoel Antônio de Paiva Neto, Michael T Lawton, and Feres Chaddad-Neto. 2018. Clinical application of an open-source 3D volume rendering software to neurosurgical approaches. *World neurosurgery* 110 (2018), e864–e872.
- [8] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. 1988. Volume rendering. *ACM Siggraph Computer Graphics* 22, 4 (1988), 65–74.
- [9] Elliot K Fishman, Derek R Ney, David G Heath, Frank M Corl, Karen M Horton, and Pamela T Johnson. 2006. Volume rendering versus maximum intensity projection in CT angiography: what works best, when, and why. *Radiographics* 26, 3 (2006), 905–922.
- [10] Daniel Haak, Charles-E Page, and Thomas M Deserno. 2016. A survey of DICOM viewer software to integrate clinical research and medical imaging. *Journal of digital imaging* 29, 2 (2016), 206–215.
- [11] M.A. Hicks. 2020. Symmetry DICOM. (2020). [https://play.google.com/store/apps/details?id=org.mahicks.simulacrum.symmetryhl=en\\_US](https://play.google.com/store/apps/details?id=org.mahicks.simulacrum.symmetryhl=en_US).
- [12] Parallax Innovations. 2018. MicroViews. (2018). <http://www.parallax-innovations.com/microview.html>.
- [13] Deepak K Jha, Puspinder Khera, Suryanarayanan Bhaskar, and Mayank Garg. 2019. Three-Dimensional Volume Rendering: An Underutilized Tool in Neurosurgery. *World neurosurgery* 130 (2019), 485–492.
- [14] James T Kajiya and Brian P Von Herzen. 1984. Ray tracing volume densities. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 165–174.
- [15] Jens Kruger and Rüdiger Westermann. 2003. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization, 2003. VIS 2003*. IEEE, 287–292.
- [16] Marc Levoy. 1988. Display of surfaces from volume data. *IEEE Computer graphics and Applications* 8, 3 (1988), 29–37.
- [17] Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.
- [18] mbits imaging GmbH. 2020. mRay - DICOM Viewer. (2020). [https://play.google.com/store/apps/details?id=org.meshl=en\\_US](https://play.google.com/store/apps/details?id=org.meshl=en_US).
- [19] Simone Perandini, N Faccioli, A Zaccarella, TJ Re, and R Pozzi Mucelli. 2010. The diagnostic contribution of CT volumetric rendering techniques in routine practice. *The Indian journal of radiology & imaging* 20, 2 (2010), 92.
- [20] Thomas Porter and Tom Duff. 1984. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 253–259.
- [21] PostDICOM. 2020. PostDICOM. (2020). <https://www.postdicom.com/>.
- [22] Francis Klumb Nicolas Roudit. Weasis: a free web-based viewer aimed for telemedicine and general practitioners.[En línea]. *Rencontres Mondiales du Logiciel Libre, sf Disponible en: http://2011.rmll.info/IMG/pdf/weasis.pdf.(Consultado el 24 de Noviembre de 2011) (????)*.
- [23] Antoine Rosset, Luca Spadola, and Osman Ratib. 2004. OsiriX: an open-source software for navigating in multidimensional DICOM images. *Journal of digital imaging* 17, 3 (2004), 205–216.
- [24] Curtis T Rueden, Johannes Schindelin, Mark C Hiner, Barry E DeZonia, Alison E Walter, Ellen T Arena, and Kevin W Eliceiri. 2017. ImageJ2: ImageJ for the next generation of scientific image data. *BMC bioinformatics* 18, 1 (2017), 529.
- [25] Open Source. 2020. MIPAV. (2020). <https://mipav.cit.nih.gov/>.
- [26] Startm. 2020. DroidRender - 3D DICOM viewer. (2020). [https://play.google.com/store/apps/details?id=com.luolai.droidrenderhl=en\\_US](https://play.google.com/store/apps/details?id=com.luolai.droidrenderhl=en_US).
- [27] Qi Zhang, Roy Eagleson, and Terry M Peters. 2011. Volume visualization: a technical overview with a focus on medical applications. *Journal of digital imaging* 24, 4 (2011), 640–664.

## Author Biography

Menghe Zhang is pursuing a Ph.D. degree in the fields of augmented reality and immersive visualization at the University of California in Dr. Schulze's Immersive Visualization Laboratory.

Dr. Jurgen Schulze is an Associate Research Scientist at UCSD's Qualcomm Institute, and an Associate Adjunct Professor in the computer science department, where he teaches computer graphics and virtual reality. His research interests include applications for virtual and augmented reality systems, 3D human-computer interaction, and medical data visualization. He holds an M.S. degree from the University of Massachusetts and a Ph.D. from the University of Stuttgart (Germany).