

# Interactive Remote Rendering

D.M. Neven

Delft University of Technology



# Interactive Remote Rendering

By

**D.M. Neven**

in partial fulfilment of the requirements for the degree of

**Master of Science**  
in Computer Science

at the Delft University of Technology,  
to be defended publicly on Friday August 22, 2014 at 13:00.

Supervisor:	Prof. Dr. E. Eisemann,	faculty EEMCS, TU Delft
Thesis committee:	Dr. Ir. R. Bidarra,	faculty EEMCS, TU Delft
	Dr. Ir. E.A. Hendriks	faculty EEMCS, TU Delft
	Ir. T. Kroes	faculty EEMCS, TU Delft

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.





## **Preface**

You are about to read my master's thesis. In this work I present the result of over a year of hard work. The subject of the project is called interactive remote rendering. The choice of this subject has to do with the accessibility of complex rendering algorithms to the patient. The opportunity to help with the design and implementation of such a system was appealing to me.

I would like to thank some people who helped me during the project. First of all I want to thank Thomas, you were such a great help to me, you explained a lot about Exposure Render and other techniques and were a real active force in the whole process of designing and implementing the system. I also want to thank Elmar for your supervision and new ideas about this project. The opportunity to go to the Hannover Messe to display our project, is something that I appreciate greatly. Thank you Rafael for helping and supporting me, especially at the beginning of the project.

Finally I want to thank you for taking the time to read my thesis. I hope you will enjoy reading my thesis.

Matthijs Neven  
Delft, the Netherlands  
August 18, 2014



# Contents

Preface .....	v
List of figures .....	ix
Abstract .....	11
1 Introduction and motivation .....	13
2 Related work .....	14
2.1 Image-based approach .....	14
2.2 Model-based approach .....	15
2.3 Conclusion .....	16
3 Technical background .....	17
3.1 Volume rendering .....	17
3.1.1 Ray tracing .....	17
3.1.1 Transfer function .....	18
3.1.2 Exposure render .....	18
3.2 Visualization techniques .....	18
3.2.1 Slicing .....	18
3.2.1 Clipping .....	19
3.3 Statistics .....	19
4 Overview .....	20
4.1 Components .....	20
4.2 Uploading .....	22
5 Implementation .....	23
5.1 Workload distribution .....	23
5.1.1 Multi-client error-based work redistribution on one server .....	23
5.1.2 Multi-server rendering .....	25
5.2 Uploading data sets .....	29
5.3 Interface .....	29
5.4 Visualization techniques and render settings .....	30
5.4.1 Clipping .....	30
5.4.2 Slicing .....	30
5.4.1 Transfer function .....	31
5.5 Interaction .....	31
5.6 Implementation details .....	32
6 User study .....	33
6.1 Comfort level .....	33
6.1.1 Setup .....	33
6.1.2 Results .....	34
6.1.3 Validation study .....	35

6.2	Viewpoint.....	35
6.2.1	Setup.....	36
6.2.2	Implementation .....	36
6.2.3	Results .....	37
7	Discussion .....	39
8	Conclusion and future work.....	40
8.1	Stream encoder.....	40
8.2	Compositor.....	40
8.3	Extra features.....	41
9	Bibliography.....	42



## List of figures

Figure 1: Classify remote rendering and remote visualization systems based on data types [1] .....	14
Figure 2: Ray tracing technique .....	17
Figure 3: slice visualization, on the left one 2D slice, on the right three slices each along one spatial dimension .....	19
Figure 4: the use of a 2D clipping plane, to show the inside of the 3D volume .....	19
Figure 5: System overview .....	20
Figure 6: Screenshot of the interface; the model setting window for clipping is shown .....	22
Figure 7: The three figures above show a snapshot of a render of the same data set at three different frames: 3 <sup>rd</sup> , 8 <sup>th</sup> and 50 <sup>th</sup> frame. Note the large amount of noise in the first image and the (almost) prefect integrated figure of the 50 <sup>th</sup> frame .....	23
Figure 8: The corresponding errors of the figures shown in are given in the chart above. This chart shows the fast decay at the beginning of the rendering and slow convergence towards the end .....	24
Figure 9: Snapshot of the work scheduling of the render nodes. Left three columns are sessions with two concurrent users, and the right four columns with four concurrent users. The colors indicated the different render nodes, and the right column shows the iteration number for that specific render node .....	25
Figure 10: System overview of the compositing setup. As an example three render nodes are displayed, but it can be any arbitrary number. The render nodes send every frame to the compositor which combines it into one image .....	26
Figure 11: This chart shows the error decay for 3 different setups: 1, 2 and 4 render nodes. As can be seen, the extra nodes result in a significant decrease of error in the beginning of the integration .....	27
Figure 12: Three snapshots during rendering. The left images are the result of a single render node, the right images are the result of four render nodes. From top to bottom: 1 <sup>st</sup> , 4 <sup>th</sup> and 10 <sup>th</sup> frame.....	28
Figure 13: Sagittal, coronal and transverse plane .....	29
Figure 14: Clipping settings; a clipping box can be set, using the three sliders .....	30
Figure 15: Clipped visualization of the Manix data set, clipped along the X-axis .....	30
Figure 16: Slicing using transfer function colors .....	31
Figure 17: Transfer function editor .....	31
Figure 18: Manix data set is used for comfort level user study .....	34
Figure 19: Settings of comfort level user study.....	34
Figure 20: The colors above indicate the colors of the transfer function. The left images correspond to the color of the bone, and the right images to the veins and arteries. The upper part represent the original coloring, the lower part the interpolated more comfortable color.....	34
Figure 21: Left image is the most uncomfortable setting, compared to the right, which is the most comfortable setting .....	35
Figure 22: Vix data set.....	35
Figure 23: Aneurix data set.....	36
Figure 24: Highlighted items in the data set. The pelvis on the left and the knee cap on the right.....	36
Figure 25: Clustering dendrogram example, clearly can be seen that sample 1 is an outlier .....	37
Figure 26: Interpolating cluster centroids using spline interpolation. The left part of the image is generated by using unordered points compared to the right part using order points .....	38
Figure 27: Six viewpoints clusters for the pelvis, extracted from user study data.....	38
Figure 28: Compositor combines the output of four render nodes into one composed image.....	41



## **Abstract**

In this thesis, we present an interactive remote volume-rendering system. Our system tries to bridge the gap between complex medical volume data and the patient. The user can easily upload their own data, which they receive from the clinician after a scan, which they can then visualize. Our application does not require any medical or volume visualization background, and can run from any device with a working internet connection and an internet browser. We also introduce a technique to use multiple machines in order to combine the computing power into one very fast renderer and the other way around: multiple concurrent users on one machine by distributing the workload efficiently.



# 1 Introduction and motivation

3D medical images from Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) are typically used to acquire a patient's anatomy and can be used for volume visualization. These images are often employed in clinical practice by doctors to perform diagnosis, plan surgeries and monitor the outcome of medical interventions. These scans produce sets of parallel slices that form a volumetric data set. Clinicians typically explore this type of data by analyzing 2D slices. Given their knowledge in human anatomy, and experience in perceiving these kind of images, they construct a mental 3D image easily. Patients often lack this ability due to a lack of experience. Nonetheless, it is becoming more common that patients receive a copy of their own medical data. However in most cases the data is of no use to them, because the tools that accompany the data are typically aimed at clinicians and do not provide easily-interpretable 3D representations of the images.

The goal of the proposed system is to bridge the gap between complex medical volume data and patients. One of the most important requirements of the system is accessibility; everyone with access to the internet must be able to use this system. Therefore, we developed an easily accessible online volume visualization system, which allows patients to explore their own medical volume data without any medical or volume visualization background.

A major challenge in making medical volume visualization accessible to the greater public is the need for powerful hard- and software. Our solution is to delegate all volume visualization tasks to a dedicated server that provides an online visualization, independent on the capacities of the patient's device. The data set stays at the server, only the images are send over to the patient. Hereby the generally large data sets do not have to be send over to the client, which saves a lot of network traffic. Moreover, the original data set stay confidential on the server, which is, due to the privacy, a requirement of such a system.

The system we propose can be used for a wide range of applications. First, doctor-patient communication can be simplified. The conversation can be done remotely if a physical meeting is not required. The patient and doctor log into the system, the doctor loads the data set and can interact or change visualization settings. The patient can see the same real-time streamed output.

More and more patients receive their volumetric data of the CT or MRI scan from the hospital on a CD or usb-stick. The system is designed to upload and visualize these personal medical scans. The patient can upload their own data to the server and can fully interact with the data set to get an insight of his or her own scan. For example, the patients can show their renders to friends and family to explain what the doctor told them.

Since the system produces photo-realistic images, the system also can be used for anatomy education. If the parameters are set correctly, a realistic anatomical insight can be given. For example the body can be "cut" open in order to visualize internal organs which are normally occluded.

This project was initiated and executed inside the group of Computer Graphics and Visualization at the TU Delft. PhD student Thomas Kroes and Matthijs Neven worked together on this project under the supervision of Prof. Dr. Elmar Eisemann.

This thesis describes the design and implementation of this interactive remote rendering system. Section 2 starts the thesis with previous and related work, followed by some technical background in Section 3. An overview of the architecture is described in Section 4, the next section contains some implementation details. Section 6 describes two user studies, which made use of the proposed system. The final sections contain the discussion and concludes the thesis and discusses the future work.

## 2 Related work

In the past years, a lot of remote rendering techniques are presented. These techniques try to overcome the limitations of lightweight devices (e.g. laptops, tablets and other handheld devices) or the lack of hard or software to render 3D data sets. A workstation with enough resources is use as a server. However, issues like bandwidth limitation, large 3D data sets, confidentiality of the data and the lack of sufficient computing power, have to be overcome. Those issues can be tackled in various ways, because different applications can have other priorities. The approaches can be classified into two major categories: model-based and image-based approach, see Figure 1. A remote visualization system that sends complete 3D data sets to the client is a model-based approach. The client needs sufficient hardware to render the 3D data sets. In an image-based approach, the server does all the actual rendering and sends images to the client. In the next sections, some examples of the two approaches are described.

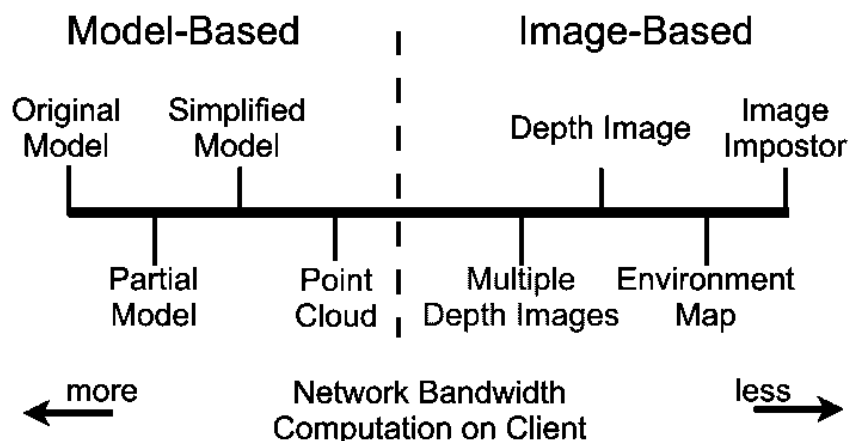


Figure 1: Classify remote rendering and remote visualization systems based on data types [1]

### 2.1 Image-based approach

Image-based remote rendering systems render all the 3D data on the server and send the images to the client, therefore the client does not need any graphical hardware. The client only displays the received images and send the interaction requests back to the server.

ParaViewWeb [1] is such a remote rendering system, based on the open source parallel data visualization framework ParaView. This approach makes use of a server-side rendering technique, provided by the ParaView engine. The images are sent to a client application, which can be JavaScript, Java or Flash-based. In [2] a similar system is presented.

An architecture that can be used for real-time exploration and management of large medical data sets was presented in [3]. The prototype system shows that it is possible to use mobile devices as remote interfaces for exploration of large volumetric medical data sets. During interaction the system limits the resolution in order to maintain a frame rate of 30 fps. After one second the client receives the full-resolution image. In case of medical data, lossy compression is forbidden, because changes in the frequency domain may affect the overall clinician's diagnosis [3]. The system is implemented as a JAVA application for Android OS, which does not meet our requirement of hardware, software and plugins independency. Furthermore it states that lossy compression in medical visualization is forbidden, because it can influence the clinician's diagnosis. However, our system features slice planes, which visualizes the data exactly, without any interpretation. The lossy compression can be overcome by only using P-frames during streaming.

The image-based approach can be adjusted in order to reduce the interaction latency; i.e. the time from the generation of user interaction request till the appearance of the first updated frame on the client. In [4]

environment maps are used to simplify the background objects. A partial panoramic, based on the viewpoint and last movement of the camera, is generated by the server. The server warps the image's coordinates into cylindrical coordinates and streams the warped images to the client, which will use a simple translation to build the partial panorama for the user. The idea is to have all possible viewpoints to which the user can go, available on the client. The buffer will be filled with images according to the camera position and direction in the virtual environment. This allows the user to freely pan the virtual camera without further assistance from the server. The interaction latency for panning only depends on the time of running view projection on the client. For other type of interactions, the server have to send new environment maps. This solution improves the latency significantly, when a complex 3D scene is used. However, in our application, only one object is present in the data: a part of the human body. Furthermore, our application makes use of a progressive renderer, which means it is not trivial to render multiple images during the rendering of the current viewpoint.

Depth images can also be used to improve the interaction latency. In [5] the server not only send color images, but also depth images to the client. The client can display the present color images if the viewpoint stay unchanged, otherwise the client can synthesize the displayed image from the depth image using 3D image warping. This technique generates novel views from the color image by performing a per-pixel reprojection, based on the depth image. Now the interaction latency is reduced to the time of image synthesis on the client, which is independent of the network. However, image warping can reduce the image quality, due to missing image information. In [6] the previous technique is extended to multiple depth images, in order to increase the quality of the synthesized images. The server renders from multiple, carefully selected viewpoints and sends the depth images to the client. This extra information allows the client to create high images when the viewpoint is changed. This is at the cost of extra computation on the server and more network traffic for streaming the extra depth images. Different techniques are presented to reduce the amount of generated depth images to save bandwidth.

Overall, the image-based approach is a good solution for our application; the data set is kept on the server, which is important with respect to confidentiality. The user also does not need powerful hardware to visualize the data. However, for a remote rendering framework the problem of interaction latency arise. As can be read above, 3D warping can partly solve this problem, but in our application we use a progressive rendering framework, which will be explained in Section 3.1. Second, 3D warping techniques are based on depth images, which are not trivial for semi-transparent volume data sets.

## 2.2 Model-based approach

In the model-based approaches, the complete 3D data set is sent to the client. As a consequence, the client needs sufficient hardware for rendering the 3D data set. Marion et al. [7] present an architecture of a remote scientific visualization system, which is based on WebGL and WebSocket technology. The system focuses on collaborative data visualization and implements it by distributing the visualization states via a web socket server. The interaction is done by one user on a master page and can be shown on the spectator page. First, the spectators receive the complete data set on their local machine, after which they receive updates about the visualization states and render the data according to the current state of the master page. The reactivity and fluidity of the system are good, for the web socket protocol can handle a lot of messages per second.

In [8] a remote volume renderer, based on the volume ray-casting algorithm, is presented. All the graphical computations are done on the client side, using WebGL. The need for specially customized hardware or plugins is eliminated for WebGL is present in current main web browsers. However, tests (in 2011) show that portable devices are not suitable to render medical volumes in real time and interactively. Noguera et al. introduces in [9] a novel technique to overcome the limitations of volume-rendering on mobile devices and WebGL. This technique is able to render up to  $512^3$  data set, without decreasing the render speed compared to the comparative technique using  $128^3$  data sets. The technique also works for WebGL, because WebGL shares at least the same limitations that mobile devices have. Hu et al. present a web-based remote laboratory in [10]. The system allows users to perform remote experiments in a virtual environment. The laboratory setup is modeled in 3D and reconstructed in a web-based interface using Flash 3D engines. The user can interact freely, and can do

similar experiments as in a hands-on laboratory. During the experiments, the 3D models are synchronized with the real laboratory setup. The experiment can be watched by different users from any angle, because the entire 3D model is continually synchronized with all the clients.

The solutions described above are sending the complete 3D data set to the client, which renders the images. This is useful for applications that require heavy computation in generating rather than rendering the 3D data sets. However, some modification on this approach can be done, in order to make the workload distribution more efficient. In some situations only a subset of the original data set is actually needed. In [11] a remote walk-through system is presented. The 3D data set is divided into zones, the zone which is directly viewable is transmitted at full resolution; the zones close to the directly viewable zone can be sent at reduced resolution; the other zones can be transmitted in a later point of time, when the network bandwidth is sufficient. Compared to the previous solutions, the start time can be reduced, due to the smaller sub data sets. However, the server has to perform extra computations to divide the data set into different zones.

In [12] the workload of rendering is distributed over both the server and client. The client generates new views by extrapolating, based on the locally available data. This data contains previous images, camera position and range data. The server needs to transmit corrections (difference images between the extrapolation and the exact view) to the client, because the extrapolation introduces small errors. These difference images are better compressible than the normal color images, and therefore reduce the network traffic. Moreover, the server does not need to correct the extrapolation every frame, but at a lower frequency. The latency is only dependent on the locally executed extrapolation.

Pajak et al. [13] present a technique developed for remote rendering in order to better balance the server/client workload. The approach improves the current methods, by making use of the computing power of the client machine combined with an image-based remote rendering system. The problem of image-based rendering is the scalability with respect to the number of clients. Their solution makes use of an augmented stream containing supplementary information, e.g. depth and motion, to reduce transfer costs. This information is used to produce a high-resolution image on the client side from a low-resolution frame rendered on the server. Hereby, the server workload and bandwidth is reduced and the auxiliary information can be used for stereo vision or various other client-side applications. As earlier said, depth images are not trivial to render for semi-transparent volume data sets.

For all the solutions described above, the data set has to be (partly) transmitted to the client, which may cost a lot of traffic and in the medical case, you often do not want to share data.

## 2.3 Conclusion

Our system is not the first to provide remote rendering for complex volume data. Still we developed our own system for several reasons. First, there does not exist a complete framework to upload and render patient medical data. Second, there is no realistic rendering approach, which helps average users to better understand the information, while we can benefit from all characteristics of Exposure Render. Third, most server-side approaches are not implemented in HTML5 and JavaScript and often dependent on specialized plugins or other software, while our solution runs off the shelf. The client side approaches do not meet our requirement of confidentiality, for the complete data set is first transferred to the client.

For our system architecture we use the setup of Wessels et al. [14], which describe a remote visualization system architecture using web sockets. This architecture makes use of modern browser techniques, i.e. HTML5.

Finally, the low hardware bar of our system allows us to record interaction information from a large set of users, which we will harvest for the purpose of developing more effective visualization schemes that render medical data more accessible to the greater public.



## 3 Technical background

### 3.1 Volume rendering

3D medical images are often used in clinical practice [15]. In general, a 3D representation of a 3D spatial region has more significance than 2D images of the same region acquired by methods, such as x-ray and ultrasound. Direct Volume Rendering techniques visualize the volumetric data set directly without changing its representation, e.g. extracting the surface of the model. Ray tracing is a technique commonly used in direct volume-rendering. This is used for complex light interaction and is based on the idea that you can model reflection and refraction by following the path of the light.

#### 3.1.1 Ray tracing

Ray tracing in volume-rendering makes use of a so-called transfer function which basically defines the optical properties, like color, opacity and glossiness. Each scalar value in the 3D data set is mapped to those properties. The algorithm starts by tracing view rays, which begin at the camera and are cast through the image plane (the pixels), into the scene, see Figure 2. Each ray has to be tested for intersection with an object. When hitting an object, three types of rays can be generated: a shadow, reflection and refraction ray. The shadow ray starts at the point of intersection and travels directly to the light source. If the ray hits another object in between, the first object will be in shadow. A reflection ray is traced in the mirrored reflection direction. Refraction rays travel through semitransparent materials and can be refracted when entering an exiting the material. By adding recursion to this algorithm, i.e. for indirect illumination, the images can be more realistic.

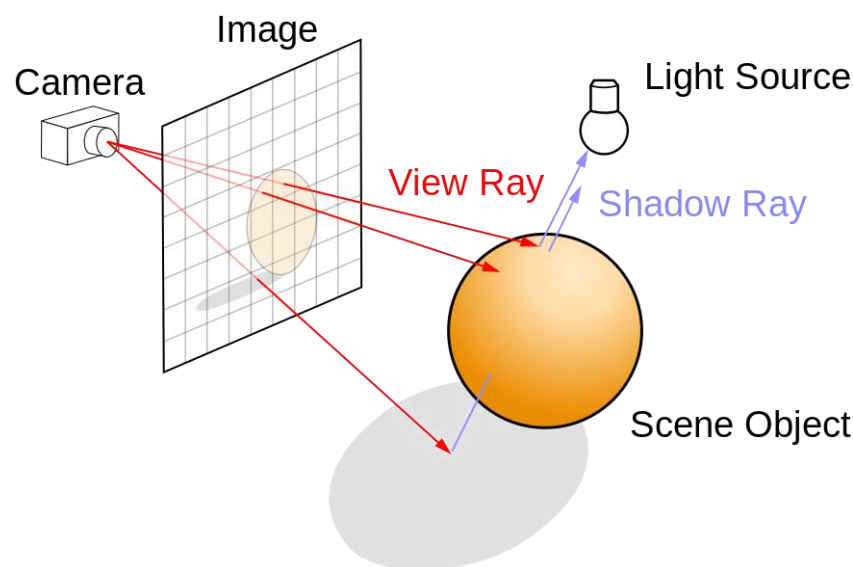


Figure 2: Ray tracing technique<sup>2</sup>

The render engine we use, Exposure Render, is based on a physically-based, unbiased rendering algorithm, i.e. physically-based renderers solve complex equations describing the light paths within a scene, unbiased means that no systematic error or bias is introduced during the rendering. The Monte Carlo Ray Tracing (MCRT) is often used to implement this type of rendering, which is based on random sampling. The statistical variance is the only source of error in the unbiased Monte Carlo algorithm. This means that multiple images of the same scene can be averaged, this would lead to a more accurate result. MCRT implements indirect illumination by integrating over all the illumination arriving at a single point on a surface. The accuracy of the images will always increase by continuing the algorithm, so a stopping condition have to be defined beforehand.

<sup>2</sup> [http://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](http://en.wikipedia.org/wiki/Ray_tracing_(graphics))

### 3.1.1 Transfer function

In our application we use a channel-based transfer function, because then you can define for each material a separate channel. For every channel the intensity range and the corresponding representation of the material (opacity, color and glossiness) can be adjusted. The intensity range defines the values in the volumetric data set, which are activated in the particular channel. The color is represented in two values, a specular and a diffuse color. Specular color defines the color of the light reflected at one angle, like a mirror, diffuse color defines the color of the light reflected in at many angles, due to the roughness of the surface.

The transfer function needs to be stored in such a way that it can be transmitted easily in a bit stream. We chose to store the transfer function in two 1D textures; one diffuse and one specular texture. These contain all the color and opacity information of all the channels combined. Since the channels can overlap, the combination of all the channels has to be done carefully. Therefore a weighted average of the separated RGB color channels is done, whereby the weights are defined by the opacity of each channel. For every intensity value the separated RGB values are multiplied by the corresponding opacity and added to the individual R,G or B-sum. The sum is then divided by the sum of all opacities for the corresponding intensity value, which result in a weighted averaged RGB value. The overall opacity value for the corresponding intensity value is defined by the maximum opacity value of the individual channels.

The bidirectional reflectance distribution function (BRDF) defines how the light is reflected at an opaque surface. The function takes the incoming and outgoing light direction with respect to the surface normal and returns the amount of reflected light. The BRDF is dependent of the material and can be controlled by parameters in the transfer function, e.g. diffuse reflection, specular reflection.

### 3.1.2 Exposure render

Exposure Render [16] is based on a MCRT framework, which integrates physically-based lighting in order to achieve photorealistic volume renderings. This framework enables simulation of an arbitrary number of shaped and textured lights, realistic shadows, and a realistic camera model. This results in high-quality images at interactive speed, due to a progressive buildup of the images. All scene parameters, e.g., transfer function, camera interaction and settings of the virtual light sources, can be modified interactively. The stochastic MC-based simulation of light transport enables the framework to render physically-based effects without being limited by number of lights, the shape of lights, the camera model, and so forth. Exposure Render implements a camera model which acts like a real-world camera, e.g., aperture controls depth of field, i.e., the distance in which objects appear to be sharp. This realistic camera model enables even artistic image generation and allows the user to produce photorealistic DVR in which attention can be drawn to particular regions by skilfully adjusting depth of field, camera position, transfer functions and more. Furthermore, due to its sampling approach, problems with aliasing and stepping artifacts are easily dealt with.

The images with its realistic features are rendered at interactive speed. This is important for the proposed system, since the clinicians and patients want to interact with a data set by looking at a realistic and easily interpretable visualization. One aspect of our work investigates these possibilities and we rely on crowd-sourcing to investigate the potential. Exposure Render runs on CUDA<sup>3</sup> enabled graphics hardware.

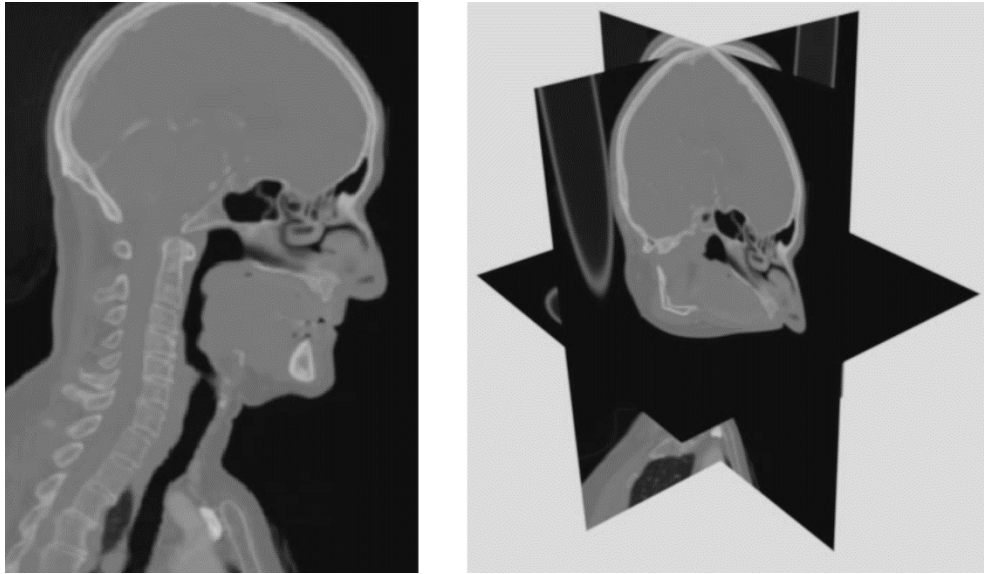
## 3.2 Visualization techniques

### 3.2.1 Slicing

Slicing entails creating a cross-section of the 3D data set. In Figure 3 two examples can be seen, the first is one 2D plane along one of the spatial dimensions, the second is one plane for each spatial dimension. Clinicians are familiar with such representations, since they are used to 2D representations like x-ray and ultrasound. Moreover this technique represent the values as they are, without any interpretation, such as a transfer function.

---

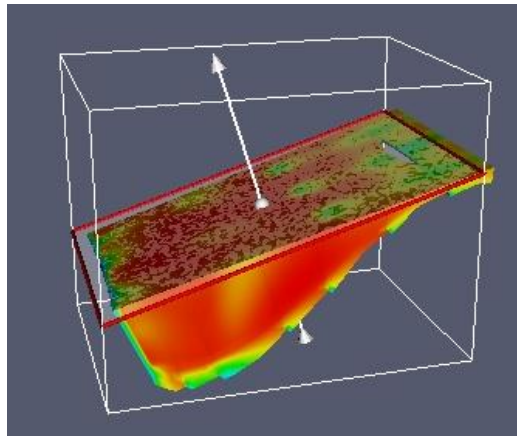
<sup>3</sup> <https://developer.nvidia.com/about-cuda>



**Figure 3: slice visualization, on the left one 2D slice, on the right three slices each along one spatial dimension**

### 3.2.1 Clipping

Clipping is a visualization technique which enables the user to cut away parts of the 3D data set. In Figure 4 a 2D clipping plane is showed, which gives an insight in the internal structure of the 3D model, without the occlusion present in the original 3D data set.



**Figure 4: the use of a 2D clipping plane, to show the inside of the 3D volume <sup>4</sup>**

## 3.3 Statistics

Interquartile range (IQR) is a measure of statistical dispersion. By ordering a sequence of values and splitting it in quartiles, the IQR can be calculated by subtracting the 1<sup>st</sup> quartile from the 3<sup>rd</sup> quartile<sup>5</sup>.

The average linkage clustering is a method for calculating distance between. The linkage function specifying the distance between two clusters, is computed as the average distance between objects from the first cluster and objects from the second cluster. The averaging is performed over all pairs (x, y) of objects, where x is an object from the first cluster, y is an object from the second cluster<sup>6</sup>.

<sup>4</sup> <http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/paraview/>

<sup>5</sup> [http://www.statistics.com/glossary&term\\_id=325](http://www.statistics.com/glossary&term_id=325)

<sup>6</sup> [http://www.statistics.com/glossary&term\\_id=714](http://www.statistics.com/glossary&term_id=714)

## 4 Overview

The system we developed is designed in such a way, that the computational expensive tasks are done on a remote server, i.e. the image-based approach. The actual volume data is kept on the server and the video stream is sent to the client, which results in no requirements of specialized hard- and software and is an important property to ensure confidentiality of the data. A photo-realistic image stream is sent to the user, who can send interaction requests back to the server. The system requires little to no prior knowledge and therefore comes with a large selection of standard settings. In our system, the patient interacts with the remote volume visualization system through an intuitive website, on which all important aspects of the volume visualization are accessible.

This results in a system which is divided into two parts, a server part and a user part (front end). The server part again consists of two sections, the server node and the render node.

By design, the system is built in such a way that the different components are replaceable. The modularity is realized by using socket connections between the components using a TCP-communication protocol. For example, the rendering engine could easily be replaced. This principle also holds for the front-end, e.g. in the first prototype version of the system, we used a dedicated C++ application to visualize the images and handle the interaction requests. Later on we wanted the system to be totally independent of any specialized hard- or software, so we replaced this application by a standardized HTML5 / JavaScript website.

Figure 5 shows the overview of the system. This diagram clearly shows the modularity of the system and the communications between the different modules.

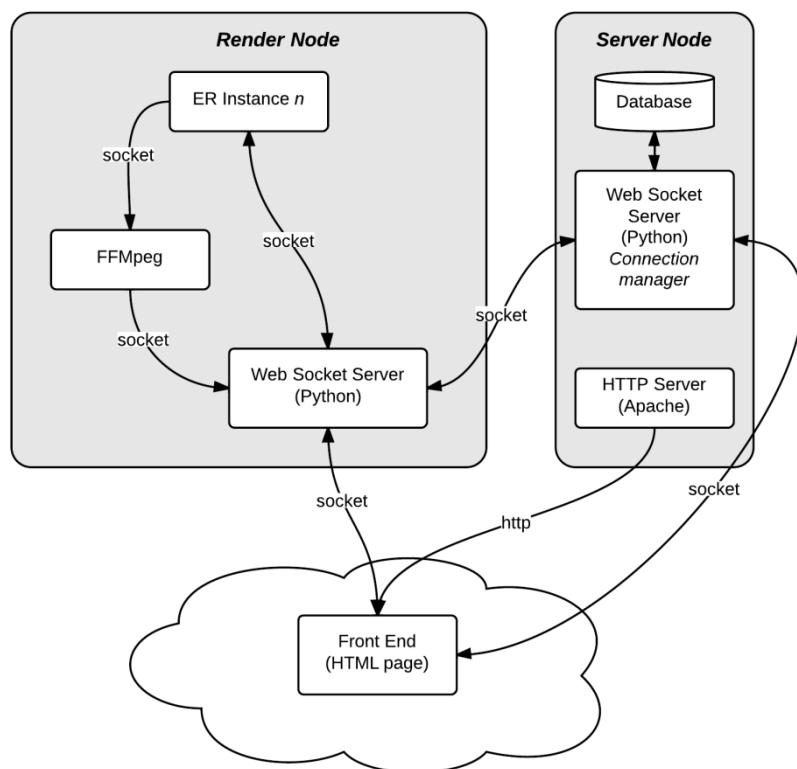


Figure 5: System overview

### 4.1 Components

This section will discuss the three components of the system: the server node, render node and the interface.

The server node is the centralized part of the system. This node manages all the connections, data storage, uploading process and user logins. After the user logs in, he or she can choose to upload their own medical data, which is explained in section 4.2, or can choose to render another available data set, their own, or a public one.

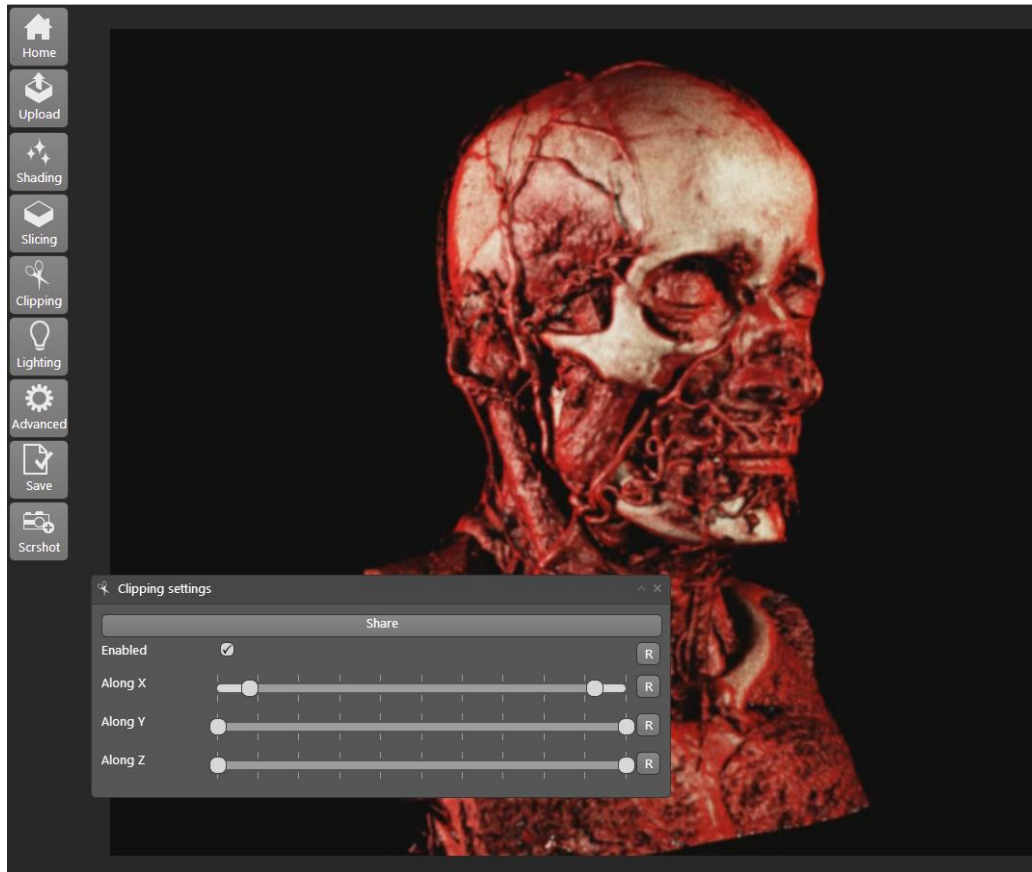
When the user select a data set to render, a render node is started. For every concurrent user a render node is running, this node can be running on a different server, and multiple nodes can be active on a servers. Different approaches of distributing the workload will be explained in Section 5.1.

The render node is responsible for producing and streaming images of the 3D data set and communicating with the front end. The render node contains an instance of Exposure Render, which produces the image stream.

Via the interface, the user can access all the features available in the system. The main item in the interface is the output screen, where the image steam is outputted on. The settings to change the appearance of the render or open the visualization techniques can be accessed via a menu on the left. This menu gives remotely access to the settings and functions of Exposure render on the server. A list of the settings are stated below:

- Shading settings: change the appearance of the data set by adjusting the transfer function;
- Lighting settings: change the amount, position, size and color of the light sources;
- Advanced settings: adjust quality parameters of the renderer;
- Slicing: access the slicing module;
- Clipping: access the clipping module;
- Save preset: save the current settings, in order to continue with the same appearance when the user visualizes the current data set again;
- Save screenshot: a screenshot of the current image will be saved, this can for example be used for sharing interesting representations of the data set.

In Figure 6 a screenshot of the interface with all the menu items can be seen.



**Figure 6: Screenshot of the interface; the model setting window for clipping is shown**

## 4.2 Uploading

The system let allow the user to upload their own volumetric data set. The standard for volumetric data in the medical domain is Digital Imaging and Communications in Medicine (DICOM). These DICOM files contain meta data about the patient: scan device, study, etc. and slice images of the 3D volume. The images can be raw RGB images or compressed with jpeg or jpeg2000. In order to make uploading data sets as simple as possible for the user, the upload process is almost automated. In Paragraph 5.2 the implementation of this process will be discussed.

After uploading, the user can choose to make their uploaded data set public to the other users. Most users do not want to be recognized on a public data set, so it is preferable that they are able to anonymize their data set. The system enables the user to clip the volume to remove for example their head or other private sections.



## 5 Implementation

### 5.1 Workload distribution

Initially the system was able to serve one user per server, but this situation is not practical, as each user would then need one dedicated machine. Therefore, we developed an algorithm to redistribute the work efficiently, in order to serve multiple concurrent users. Specifically, we concentrate on multiple users connecting to a single server, this technique is explained in Paragraph 5.1.1; multi-server rendering will be addressed in Paragraph 5.1.2.

#### 5.1.1 Multi-client error-based work redistribution on one server

A simple method to redistribute the work would be to let each render node work as fast as it can without having any notice of the other nodes on the same machine. In consequence, the same resources of the GPU are used by different render nodes. This can have a negative influence on the performance, or can even lead to memory errors. A solution is to take turn, every render node starts when the previous one finishes. If the last one finishes, the first one starts again. This solution is far from optimal, as the quality of the Monte Carlo Ray Tracer is increasing with every extra sample, i.e. the quality is related to the number of samples according the formula  $(\sqrt{n})$  [16]. In consequence, the frequency of rendering can be reduced when the number of iterations increases.

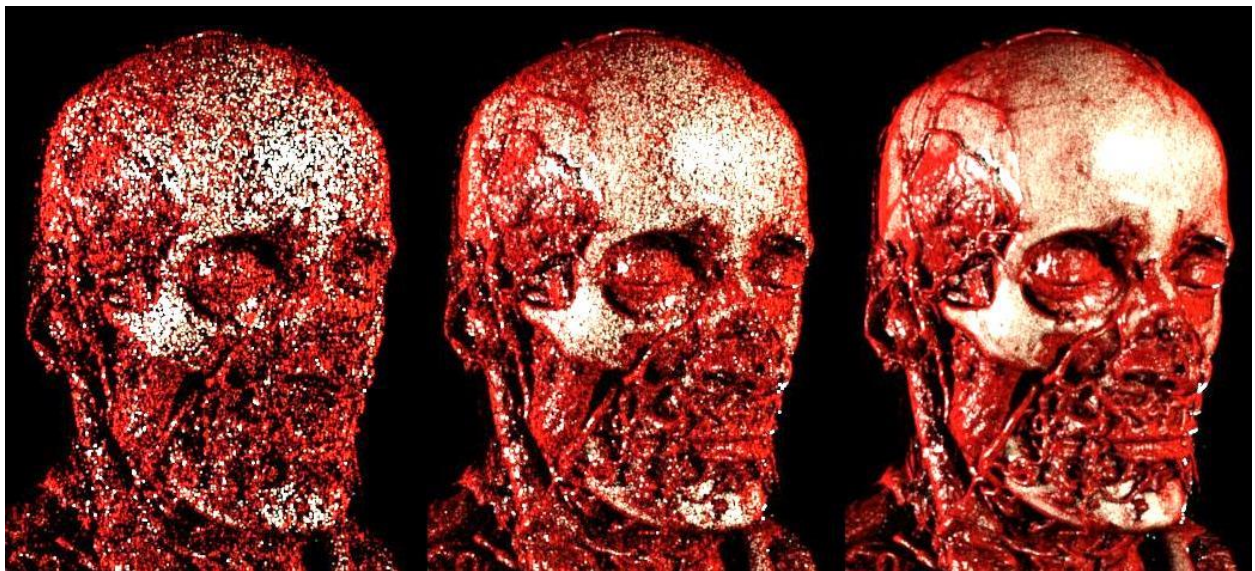
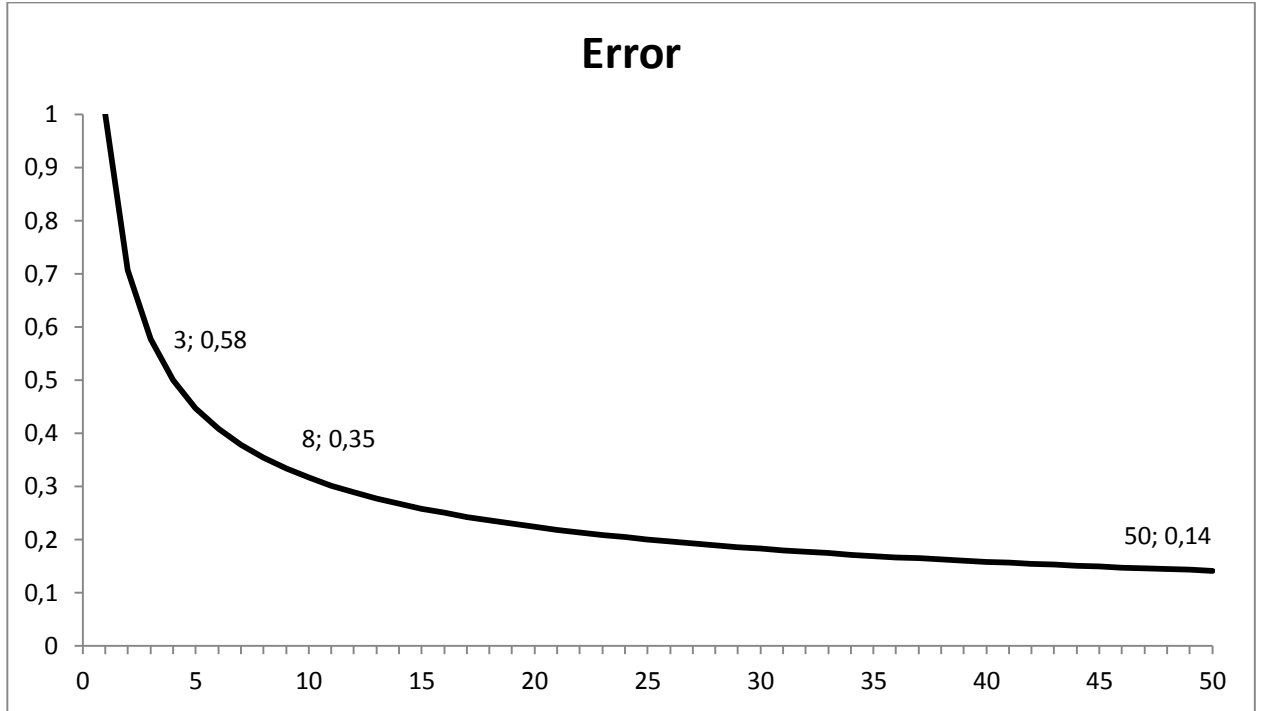


Figure 7: The three figures above show a snapshot of a render of the same data set at three different frames: 3<sup>rd</sup>, 8<sup>th</sup> and 50<sup>th</sup> frame. Note the large amount of noise in the first image and the (almost) perfect integrated figure of the 50<sup>th</sup> frame



**Figure 8:** The corresponding errors of the figures shown in are given in the chart above. This chart shows the fast decay at the beginning of the rendering and slow convergence towards the end

Figure 7 and Figure 8 show the error of a rendering at three different frames. As can be seen, the first image (3<sup>rd</sup> frame) is very noisy and needs a lot of integration steps to reduce the noise, the 50<sup>th</sup> frame is almost perfect.

Our solution is based on the expected error (noise) of the current render frames, which is estimated by  $O(\frac{1}{\sqrt{n}})$ , where  $n$  is the number of iterations. This is based on the formula of the incident light, i.e. the light that falls on a subject, either directly or indirectly,  $I_i = \frac{1}{N} \sum_{j=1}^N \frac{f_i(c_j)}{p(x_j)}$  defined in [16]. The number of iterations is reset every time an interaction request is received. The system decides which render node to execute based on the probability density function of the remaining error of the render nodes. The priority is then proportional to the error ( $\frac{1}{\sqrt{n}}$ ). This results in a high priority of render nodes with a low iteration number, i.e. the node will render significantly more samples during the starting phase of the rendering process. In consequence, the rendering process converges quickly for individual renderings, without losing the flexibility of handling multiple concurrent users.

A manager application delegates the work distribution by sending a message to the render node which have to render at that time. The render node performs one iteration and send the resulting image. After receiving the image, the server node calculates, according the mechanism above, which render node has to render next, and the process starts again from the beginning.

The result of the algorithm above is illustrated in Figure 9. All tests are done on one machine, whereby all of the individual render nodes use the same data set to visualize. First, we tested two concurrent clients, shown in the left three columns of the image. When interacting with one render node, it has priority over the second one. As said, the distribution depends on the amount of estimated error, which results in very few render iterations of node two in the two left sessions. The rendering workload is more evenly distributed when both nodes are interacting, as can be seen in the third column. The same principle holds for more render nodes; in the right four columns we used four render nodes on one machine. The similar result are clearly visible, especially the right-most column is interesting; the session was a very extreme situation, i.e. a constant interaction of all four render nodes. Here, every render node needs maximum computing power of the machine, as the error of all nodes after



interaction is relatively high. In theory, in this situation, the convergence is a factor corresponding to the amount of concurrent users slower, compared to one concurrent user per machine. However this is worst case scenario, i.e. it is unlikely that all the nodes receive interaction requests continuously. In this scenario, the algorithm acts like the take turn approach, but in any other case it handles the work load much more efficiently.

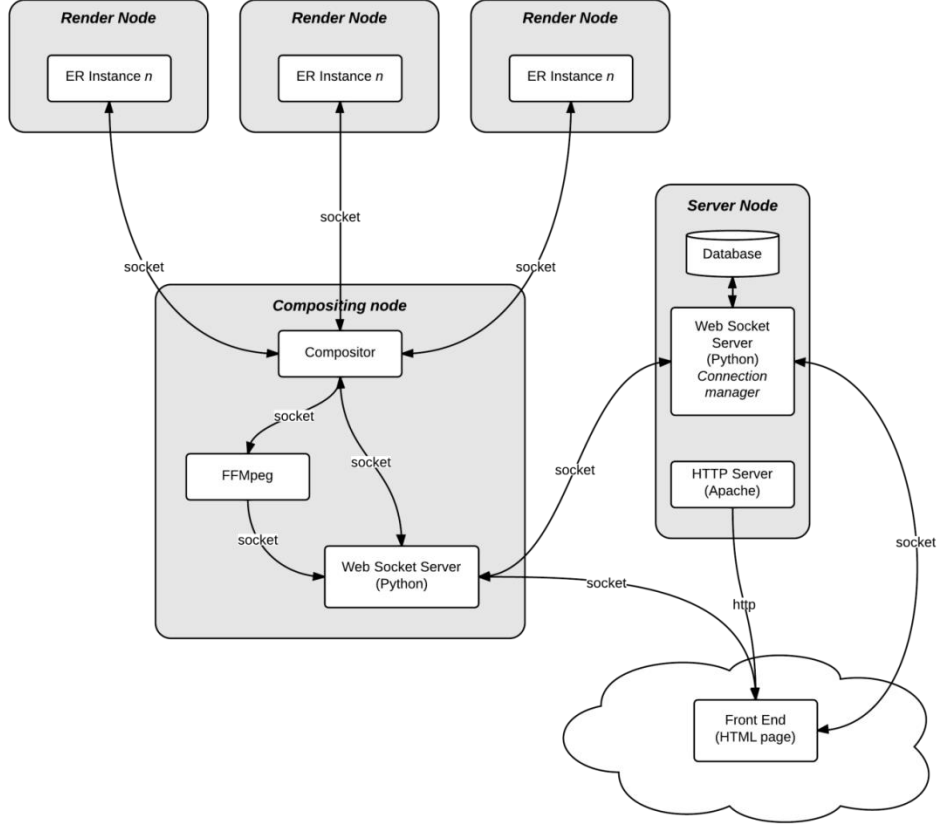
2 render nodes			4 render nodes			
1 1	1 1	1 1	1 1	1 1	1 1	3 1
1 2	1 2	2 77	1 2	2 45	1 2	3 2
1 3	1 3	1 2	2 804	1 2	1 3	4 14
1 4	1 4	1 3	1 3	2 46	1 4	1 8
1 5	1 5	1 4	2 805	3 29	1 5	4 15
1 6	1 6	1 5	1 4	1 3	1 6	1 9
2 851	1 7	1 6	1 5	1 1	1 7	3 3
1 7	1 8	1 7	1 6	4 72	1 8	2 17
1 8	1 9	1 8	1 7	4 73	1 9	2 18
1 1	1 10	2 78	1 1	4 74	1 10	3 4
1 2	1 11	2 79	1 2	4 75	1 11	1 1
1 3	1 12	1 9	2 806	3 30	1 12	1 2
1 4	1 13	1 10	1 3	1 2	1 13	1 3
1 5	1 14	1 11	1 4	2 47	2 2687	1 4
1 6	1 15	1 12	1 5	1 3	2 2688	1 5
1 7	1 16	2 80	1 6	1 4	1 14	4 16
1 8	1 17	1 13	1 7	2 48	1 15	4 17
1 9	1 18	1 14	1 1	1 5	3 2595	3 5
1 10	1 19	1 15	1 2	2 49	1 16	1 6
1 11	1 20	1 16	1 3	1 6	1 17	4 18
1 12	1 21	1 17	2 807	2 50	1 18	3 1
1 13	1 22	1 18	3 682	1 7	1 19	1 7
1 14	1 23	2 81	2 808	2 51	1 20	1 8
1 15	1 24	2 82	1 4	1 8	1 21	3 2
1 16	1 25	1 19	1 5	1 9	1 22	4 1
1 17	1 26	2 83	3 683	4 76	3 2596	2 19
2 852	2 2259	2 84	1 1	1 10	3 2597	1 9
1 18	1 27	1 20	1 2	1 11	1 23	3 3
1 19	1 28	2 85	3 684	4 77	1 24	2 20
1 20	1 29	1 21	1 3	2 52	1 25	2 21
1 21	1 30	2 86	4 1024	1 12	1 26	2 22
1 22	1 31	1 22	1 4	4 78	1 27	4 2
1 23	2 2260	1 23	2 809	2 53	1 28	2 1
1 24	1 32	2 87	1 5	2 54	1 29	3 4
1 25	1 33	1 24	3 685	2 55	1 30	2 2
1 26	1 34	1 25	1 6	2 56	1 31	2 3
1 27	1 35	1 26	1 7	4 79	2 2689	3 5
1 28	1 36	1 27	1 8	3 31	1 32	3 6
1 29	1 37	1 28	1 9	3 32	1 33	4 3
2 853	1 38	1 29	4 1025	1 13	1 34	2 4

Figure 9: Snapshot of the work scheduling of the render nodes. Left three columns are sessions with two concurrent users, and the right four columns with four concurrent users. The colors indicated the different render nodes, and the right column shows the iteration number for that specific render node

In the case of multiple servers, this technique can be applied on each individual server. If a new user logs into the system, the user is redirected to the server with the least amount of concurrent users. In this way the work is distributed evenly among the different servers.

### 5.1.2 Multi-server rendering

Due to the fact that the system is modular and communicates via sockets, the render nodes on different machines can be combined to one rendering system. This is the opposite of the technique described in the previous section. Now we can combine the power of the different machines into one very fast converging renderer.

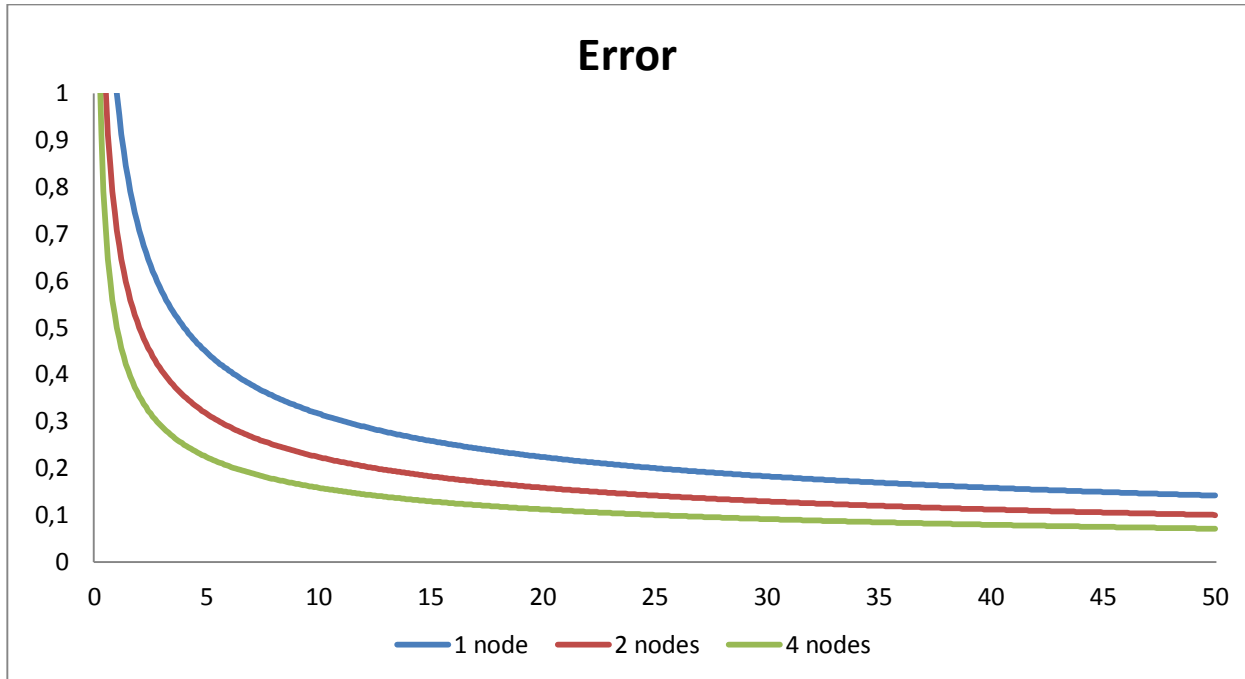


**Figure 10: System overview of the compositing setup.** As an example three render nodes are displayed, but it can be any arbitrary number. The render nodes send every frame to the compositor which combines it into one image

As said earlier, the render node makes use of the Monte Carlo Ray Tracer algorithm, which is based on random sampling. Rendering one data set with multiple render nodes, results in slightly differing images due to statistical variance. These different images are all unbiased and therefore can be combined by averaging the result images of the different render nodes.

As can be seen in Figure 10, the individual render nodes send their results to the compositing node. The compositor combines those images into one final image. The result of the compositing setup, is a speed up by the factor of  $k$  and the error will be reduced by a factor of  $\sqrt{k}$ , where  $k$  is the number of render nodes. Combining with the error of a single render node,  $O(\frac{1}{\sqrt{n}})$ , it makes the formula of the error:  $O(\frac{1}{\sqrt{nk}})$ . Figure 11 shows the error for a different number of render nodes, the significant decrease of error in the initial phase can be seen clearly in this diagram.

Due to the asynchronous implementation of the communication, ghosting can occur during camera interaction. When the user sends an interaction request, an update to all render nodes will be transmitted. These render nodes will not receive the message at exactly the same time, consequently some render nodes are not up to date. When the representation of the model is modified, initially some nodes will send images of the unmodified model, which is called ghosting. In our system we keep track of the interaction inside the compositing node; when an interaction request is received from the GUI, an iterator is increased and sent to each render node. Along with the rendered frame, the iteration number is sent back. With this number the compositor can check if the interaction on each render node is up to date. If not, the frame of the specific render node will be skipped. Now it can happen that the compositor has less frames to average, but this results in a better image than the ghosted version.



**Figure 11:** This chart shows the error decay for 3 different setups: 1, 2 and 4 render nodes. As can be seen, the extra nodes result in a significant decrease of error in the beginning of the integration

Figure 12 shows the result of the compositing algorithm at different snapshots during the starting phase of the rendering. As can be seen, the composited versions on the right have significantly less noise compared to the single-rendered ones on the left. In theory, the composed version with 4 nodes at iteration 1 should result in the same image as the single version in iteration 4. Figure 12 show that after 10 iterations most of the noise is gone, while this would take 40 iterations in the normal setup.

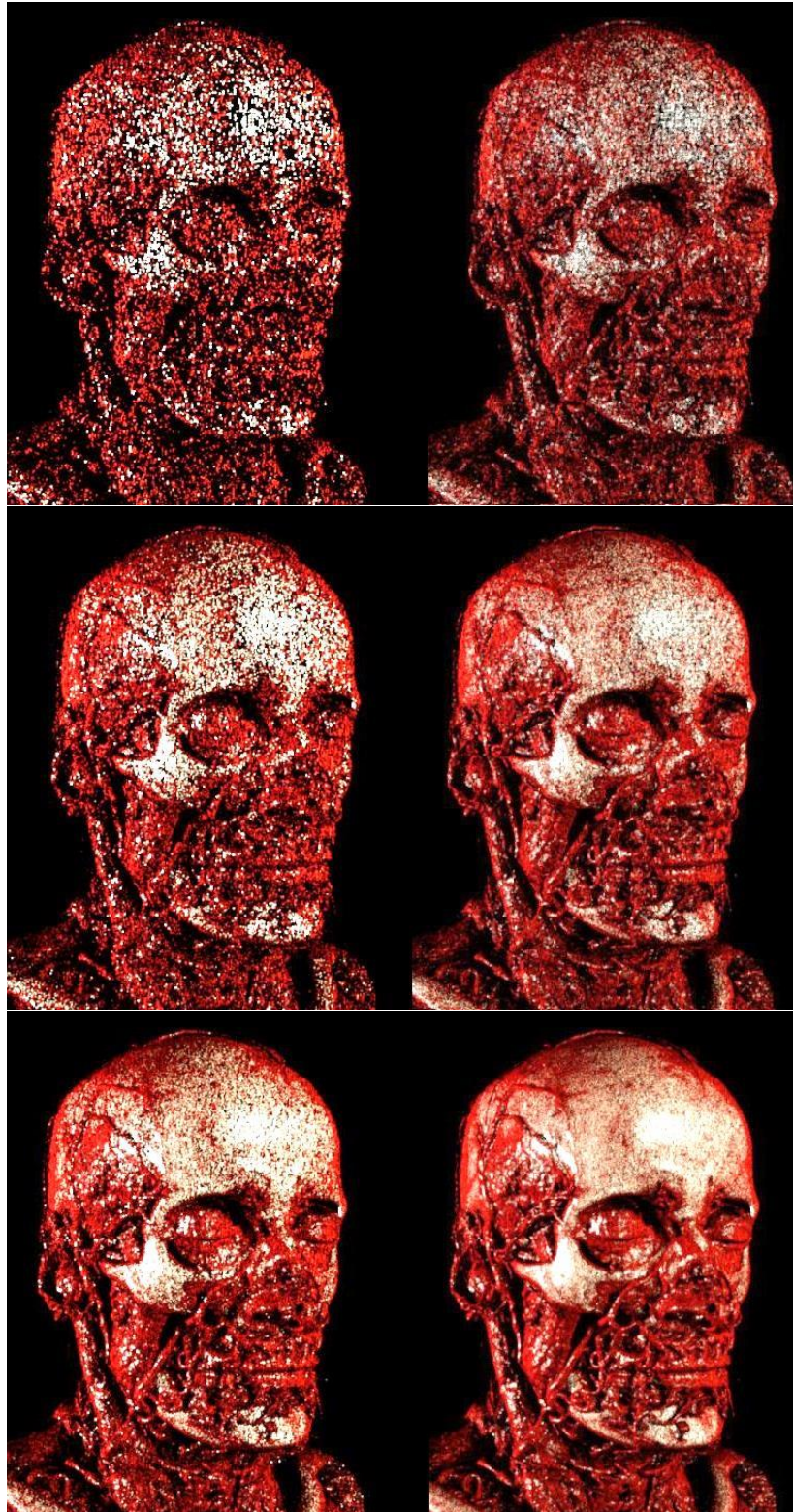


Figure 12: Three snapshots during rendering. The left images are the result of a single render node, the right images are the result of four render nodes. From top to bottom: 1<sup>st</sup>, 4<sup>th</sup> and 10<sup>th</sup> frame

## 5.2 Uploading data sets

As mentioned in Section 4.2, uploading of a data set should be very easy and intuitive. In this section we will describe how we implemented this process.

First, the user has to select the directory or device, e.g. CD or USB-stick, which is automatically scanned entirely for suitable data sets. All the containing files are checked for the DICOM tag and marked if so, after which they are categorized into separate scans. The result is a list of scans available in the directory. However, the system is not able to visualize some of those scans, e.g. time dependent series. These scans are filtered from the list and not shown to the user. Scout images are another example. Usually the output DICOM files are images in the transverse plane (see Figure 13), but sometimes extra slices (scouts) have been acquired to align the main DICOM files. These scout images are aligned according the sagittal or coronal plane and should be removed from the data set.

The two problems described above, results in the necessity of reading the metadata of the DICOM files on the client side. Every DICOM file in the selected folder is read and sorted by the series ID. For every series the origin of all slices are checked, if the origins are translated in one direction for every slice, the series is a suitable series, i.e. no time series. The scout images are removed by comparing the image positions: scouts have a different orientation than the main slices.

After all these checks, the suitable data sets are shown, after which the user can select one to upload to the server. On the server, the DICOM files are combined into a volume by using the `vtkGDCMImageReader` module. After creating the volume and some post processing the DICOM files are deleted, so the original data and all private information is deleted.

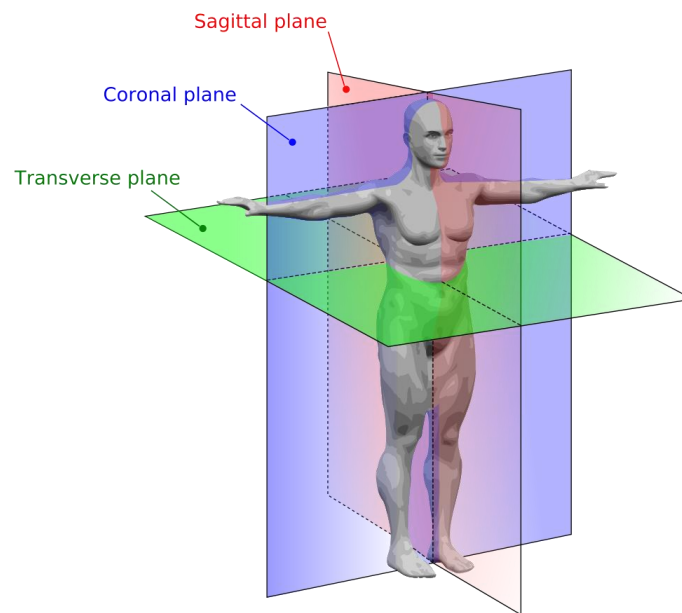


Figure 13: Sagittal, coronal and transverse plane

## 5.3 Interface

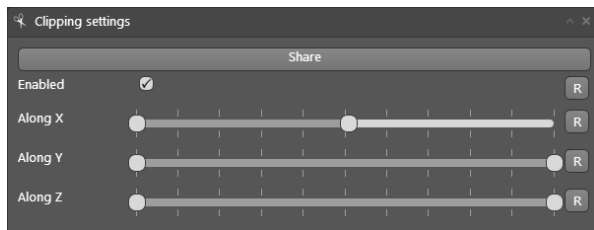
We required the interface not to be overwhelming, because the application should be intuitive and easy to use. Therefore, we placed a menu on the left, which gives access to the settings windows, as described in Section 4.1. The windows are modal, so can be dragged to any position the user wants.

The interface is built in HTML5 and JavaScript, since most of the devices have an HTML5 capable browser; even tablets and cellphones. In our application two features, which are introduced in the HTML5 standard, are used: the canvas element and directory uploading. We use the canvas element to capture the user's camera interaction. Second is the ability to process an entire directory on the front-end, instead of selecting multiple files, which we use in the uploading process. This is important, because the data set can consist of multiple files in multiple folders. Section 5.2 explains the benefit of selecting a complete directory to upload.

## 5.4 Visualization techniques and render settings

### 5.4.1 Clipping

In our system a clipping functionality is present. As can be seen in Figure 14, the clipping box can be set using three sliders, one slider per spatial dimension. Using clipping, the inner structure of a data set can be shown, without occlusion of other material, which can be seen in Figure 15. The VTK reslice module is used to clip the volume to the desired output.



**Figure 14: Clipping settings; a clipping box can be set, using the three sliders**



**Figure 15: Clipped visualization of the Manix data set, clipped along the X-axis**

### 5.4.2 Slicing

Another technique often used in render frameworks is slicing. The system can handle three slices at the same time, each along one of the three spatial dimensions, see Figure 3. The standard coloring of the slices is in grayscale, for the original values are one-dimensional. Nonetheless, an added feature in our system is the transfer color projected on the slices, the result can be seen in Figure 16. As already mentioned, slicing is especially common to clinicians, they use this visualization technique to perform diagnosis, plan surgeries and monitor the outcome of medical interventions. Therefore, it is useful to give the clinician the possibility to this visualization technique.



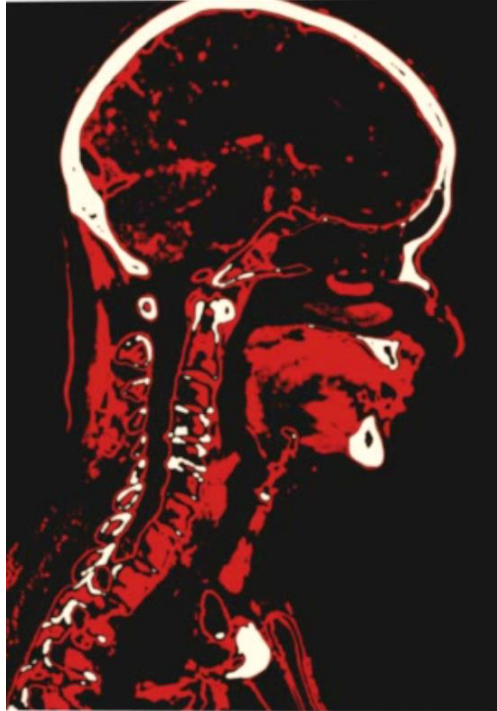


Figure 16: Slicing using transfer function colors

#### 5.4.1 Transfer function

In our application we make use of the channel-based transfer function, as described in Paragraph 3.1.1. In Figure 17 the transfer function editor is shown, where can be seen the different channels on the top and the individual settings in the rest of the window. The channels can be enabled, the intensity range on which the channel applies on and the appearance of the channel, e.g. color and opacity, can be adjusted.

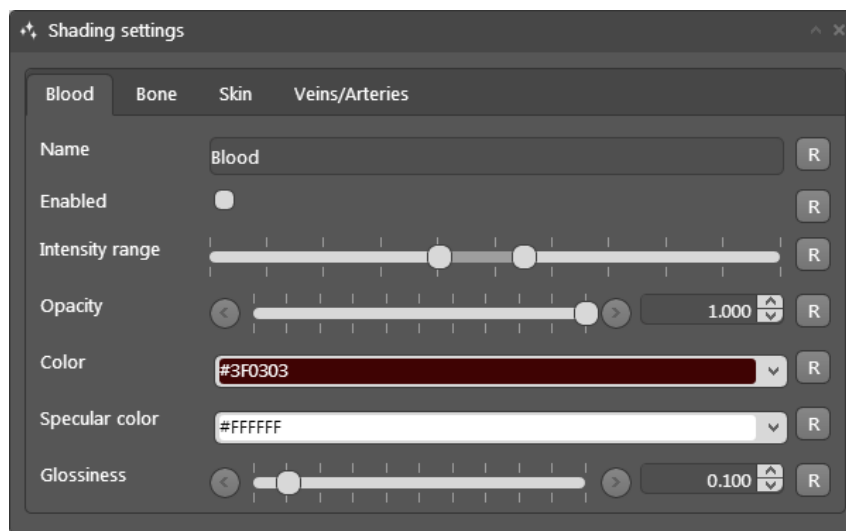


Figure 17: Transfer function editor

## 5.5 Interaction

Every data set has a corresponding preset file, encoded in XML, which contains all settings, e.g. transfer function, camera and light positions. These presets are copied from a default preset file and saved on the server node. The presets are decoded when the data set is loaded into Exposure Render, after which all render parameters are set according to the preset. At the same time the XML file is sent to the front-end HTML page via base64 encoding, where the presets are loaded into the user interface. During interaction requests, e.g.

camera interaction, transfer function editing, an XML representation in the render node is kept up to date. At every moment of the session the user can save the current preset in order to continue later on, without setting all the parameters by hand.

The XML representation can be of use with a general user study. If we want to gain information about camera interaction and settings changes of the user, the updates can be send via this XML representation. On the server information can be deduced to give an insight in the way the user interact with the system, e.g. do they make use of standard presets, or do they change individual settings.

## 5.6 Implementation details

Since the application consist of multiple modules, different languages and scripts are used. For the server and render nodes we chose to use Python, using the libraries PyQt and Tornado<sup>9</sup>. The latter is used to communicate via WebSockets with the front-end. The render engine we used, Exposure Render, is written in C++. Some modifications were needed in order to remove the GUI and attach the remote controls, implemented via communication sockets.

The HTML5 webpage is several JavaScript libraries. One of them is jqWidgets<sup>10</sup>, a jQuery<sup>11</sup> based library, which we used for building the interface. The Three.js<sup>13</sup> library has built-in camera controls which can be applied on the HTML5 canvas. The resulting camera parameters are sent to the server in order to update the model.

As explained in Section 5.2, HTML5 is used to process an entire directory. This feature is implemented by WebKit<sup>14</sup> and is called webkitdirectory. For now, this implementation is only available in a Chrome browser, so the upload module is only functional in this specific browser. All other features are functional in current main web-browsers.

The output of Exposure Render are raw RGB-images. The conversion to a stream and encoding is performed by a standard audio/video codec library: FFmpeg<sup>15</sup>. The stream of images are encoded using MPEG1 compression and transmitted via sockets to a web-socket server. This setup results in a reasonable performance regarding latency and quality. The stream is received by the front-end and is decoded by jsmpeg<sup>16</sup>, a JavaScript based MPEG1 decoder. The decoded result is displayed on the HTML5 canvas. This implementation of streaming holds the requirement of device or plugin independency, for it makes use of standard HTML and JavaScript libraries.

Since the introduction of WebSockets in the W3C standard, full-duplex communication via the JavaScript interface is accessible. This technology allows us to develop real-time synchronization between multiple users over the internet via the web browser. The message headers have been kept small and due to the persistent connection for a single user, the latency is reduced. This connection makes it possible for the server and client to push messages to each other at any given time. This behavior is essential for our application, for we are dealing with a real-time interactive bi-directional system.

---

<sup>9</sup> <http://www.tornadoweb.org/>

<sup>10</sup> <http://www.jqwidgets.com/>

<sup>11</sup> <http://jquery.com/>

<sup>13</sup> <http://threejs.org/>

<sup>14</sup> <http://www.webkit.org/>

<sup>15</sup> <http://www.ffmpeg.org/>

<sup>16</sup> <https://github.com/phoboslab/jsmpeg>



## 6 User study

The system is very well suited to perform user studies. Therefore we performed two user studies, one to investigate the influence of the color of a material on the comfort level of the user. In the other study we try to extract general view points for specific regions in a 3D data set out of crowd generated data.

### 6.1 Comfort level

The proposed system is supposed to be used by patients without a medical background, who are not used to realistic volume renders of the body. Therefore, we want to find a way to adjust the comfort level represented by one metric. To check whether color of a certain material influences the comfort level of a user, we performed a user study.


#### 6.1.1 Setup

First the reference color of bone and blood are shown on the Manix<sup>18</sup> data set as can be seen in Figure 18. The user first can indicate how comfortable they feel about the visualization. After this rating, the colors of the bone and blood can be changed, in order to make the visualization more comfortable to them.

---

<sup>18</sup> <http://www.osirix-viewer.com/datasets/>



 Change color

Inspect the volume first.

Please indicate from 0 - 10 how comfortable the visualisation appears to you.

Please change the colors in such a way that the visualization looks more comfortable.

Bone

#E8BA94

Veins/Arteries

#6D0404

Submit

**Figure 18: Manix data set is used for comfort level user study**

**Figure 19: Settings of comfort level user study**

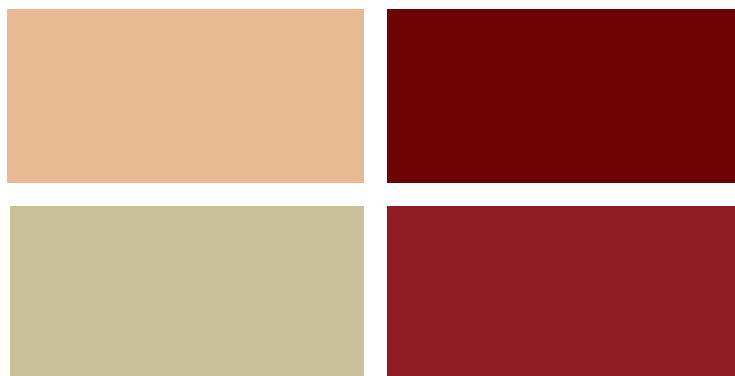
### 6.1.2 Results

In this study 43 persons participated. The results of the study have to be processed to deduce information. The samples of the participants who feel really uncomfortable have to be strongly count, therefore the samples are weighted by the inverse of the comfort rating. The results can be seen in Table 1.

Material	Measurement	Number of participants	Mean	Standard deviation
Bone	Difference	43	r: -24, g: 5, b: 7	r: 55, g: 50, b: 67
Bone	Weighted difference	43	r: -30, g: 7, b: 6	r: 84, g: 52, b: 84
Veins	Difference	43	r: 36, g: 20, b: 29	r: 62, g: 29, b: 53
Veins	Weighted difference	43	r: 34, g: 21, b: 32	r: 76, g: 33, b: 62

Table 1: Results of the comfortable level user study

Now a grossness scale can be extracted from the weighted average by linearly interpolating between the reference color and the average. The color in Figure 20 show the result of the colors of the reference setting and the more comfortable setting.



**Figure 20: The colors above indicate the colors of the transfer function. The left images correspond to the color of the bone, and the right images to the veins and arteries. The upper part represent the original coloring, the lower part the interpolated more comfortable color.**

In order to use this information in the transfer function, we linearly extrapolated the difference compared to the reference color and used these values to change the comfort level of the visualization. The result of the five

different settings can be seen in Figure 21. The images are ranged between -2 times the weighted difference and +2 times the weighted difference.



**Figure 21:** Left image is the most uncomfortable setting, compared to the right, which is the most comfortable setting

### 6.1.3 Validation study

In order to validate the results above, we performed a validation study with 16 participants. We asked them to order the images in Figure 21 from uncomfortable to comfortable. The initial order of the images was randomly chosen, to prevent any bias.

We do not make a difference between the 1<sup>st</sup> and 2<sup>nd</sup> because they are should both be less comfortable than the reference image, the same holds for the 4<sup>th</sup> and 5<sup>th</sup>. Since the data is in an ordinal scale, we can check for every participant if the 1<sup>st</sup> and 2<sup>nd</sup> are rated as less comfortable than the reference image and if the 4<sup>th</sup> and 5<sup>th</sup> are rated as more comfortable. On average 63% of the participants rated the first to images as less comfortable and 59% rated the last two images as more comfortable. Those numbers are not very convincing, so a follow up study is required to verify the earlier results.

## 6.2 Viewpoint

In general a part of the body can be visualized from different angles and distances, however most likely there exists for those regions several major viewpoints which form a representation of that specific body part. These viewpoints can for example be used for initial investigation of a data set by an inexperienced user, or a roundtrip along those viewpoints for educational purposes.

In order to find suitable viewpoints of volume-rendered medical data sets, we performed a user study. As in [17] we designed a subjective experiment to analyze users' view preferences. We hope to find so-called canonical views of different parts of the body. Canonical views have several factors which are described in [18]:

- Goodness of recognition: salience and significance of features, stability with respect to small transformations and the number of occluded features.
- Familiarity: the views that are most frequently encountered.
- Functionality: the views that are most relevant for how we use the object.
- Aesthetic criteria: geometric proportions can have influence regarding the view we prefer.



**Figure 22:** Vix data set

This user study is a proof of concept in order to show the possibility of crowd generated viewpoint selection.

### 6.2.1 Setup

The users have to choose the best view by changing the camera position. We chose to visualize the Aneurix and Vix<sup>19</sup> data set, which can be seen in Figure 22 and Figure 23. The Aneurix data set is used when defining camera positions representing specific regions and the vix data set to get camera positions for the data set as a whole. The Aneurix data set is clipped along the sagittal plane in order to prevent confusion between the left and right part of the body. The hip joint and knee are the items to choose the best view point for. In order to explain which parts they have to show, the corresponding areas are highlighted, which will be explained in Paragraph 6.2.2.

Every time a participant switches to another data set, the camera change to a randomized position, in order to keep the test unbiased.

### 6.2.2 Implementation

The regions of interest have to be highlighted in order to visualize the body part which the user have to select. This is implemented by using VTK modules in DeVIDE<sup>21</sup>. The part to be highlighted is covered by a (manually positioned) sphere created by `vtkSphereSource`, which is transformed to image data by `vtkPolyDataToImageStencil` and `vtkImageReslice`. The result of this transformation is a sphere which contains high values, i.e. the values are in the order of the maximum value of the original image. This image is added to the original using `vtkImageMathematics` to create an offset on the region of interest. The result is saved to a file which again can be loaded by the system. We can use these high values to separate the region of interest using the transfer function, i.e. we use a highlight channel which is applied to the high values in the region of interest. We use the color blue because in a realistic rendering, this color does not occur naturally. The results can be seen in Figure 24.



Figure 23: Aneurix data set

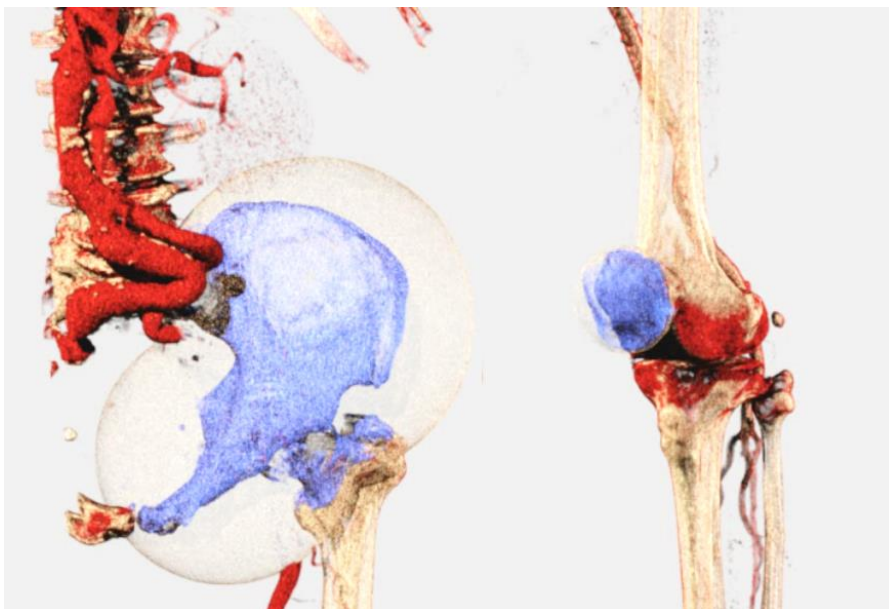


Figure 24: Highlighted items in the data set. The pelvis on the left and the knee cap on the right

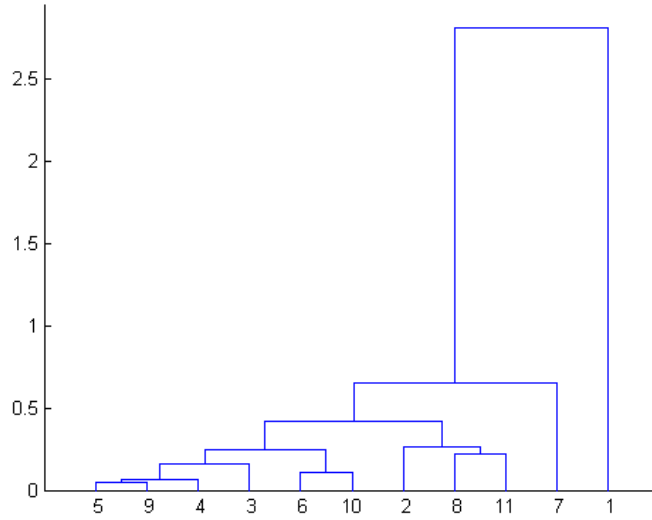
<sup>19</sup> <http://www.osirix-viewer.com/datasets/>

<sup>21</sup> <http://www.cg.its.tudelft.nl/Projects/DeVIDE>

### 6.2.3 Results

After acquiring this user study data, some post processing is needed. First outliers have to be removed, because these positions are probably made by accident. Secondly, we have to determine the amount of clusters, which will be the basis of the canonical viewpoints. The last step is connection the points into a path, i.e. a virtual roundtrip.

First, we use a combination of hierarchical clustering and an interquartile range to remove single, strong outliers. A sample is considered a single and strong outlier, if the sample is by oneself in a cluster and the distance to the other clusters is large. The bounds of the distance to the other clusters is defined by  $[first\ quartile - 3 * IQR, third\ quartile + 3 * IQR]$ . We consider the sample as a strong outlier if it is outside these bounds. This process is repeated until the stated condition is not met anymore. An example can be seen in Figure 25, where one outlier is present in the data. By only removing the single outliers, the (most probable) error samples are removed. However, the viewpoints which are selected by a few (at least more than one) are kept, because those can give an interesting variant viewpoint.



**Figure 25: Clustering dendrogram example, clearly can be seen that sample 1 is an outlier**

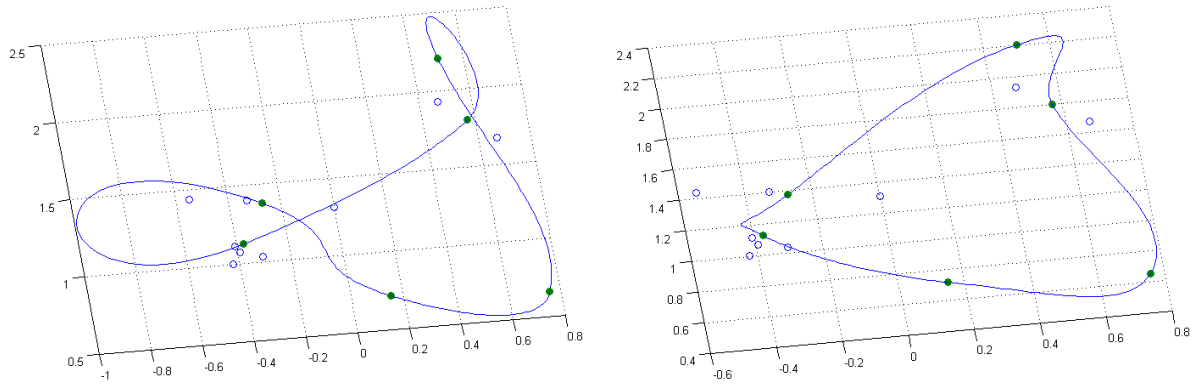
After removing the outliers, the amount of clusters have to be determined. Therefore we use a cluster evaluation method, called silhouette evaluation [19]. This criterion is composed of two scores:

- The mean distance between a sample and all other points in the same class;
- The mean distance between a sample and all other points in the next nearest cluster.

We chose to set the range of number of clusters from 1 to half the number of samples  $n$ , because we want to prevent the algorithm to stuck in the local optimum of one cluster per sample. Now, an optimal number of clusters can be chosen, using the Gaussian mixture distribution. The distribution can handle the different size and denseness of the distributions, which probably will be the case in our data: one or two high-density clusters and some low density clusters corresponding with the non-trivial viewpoints. This number is used as input for the actual clustering, again based on the Gaussian mixture distribution.

The centroids of the clusters are the basis of a virtual roundtrip around the item of interest. The path is created by interpolating the cluster centroids. But first the cluster centroids have to be ordered, to make the roundtrip smoother. We chose to order the centroids by the angle to the center (average) of the data in the  $xy$ -plane. The result of the ordering can be seen in Figure 26, the unordered centroids create a far from efficient roundtrip, compared to the smoothed trip based on the ordered centroids. We used spline interpolation introduced by [20]

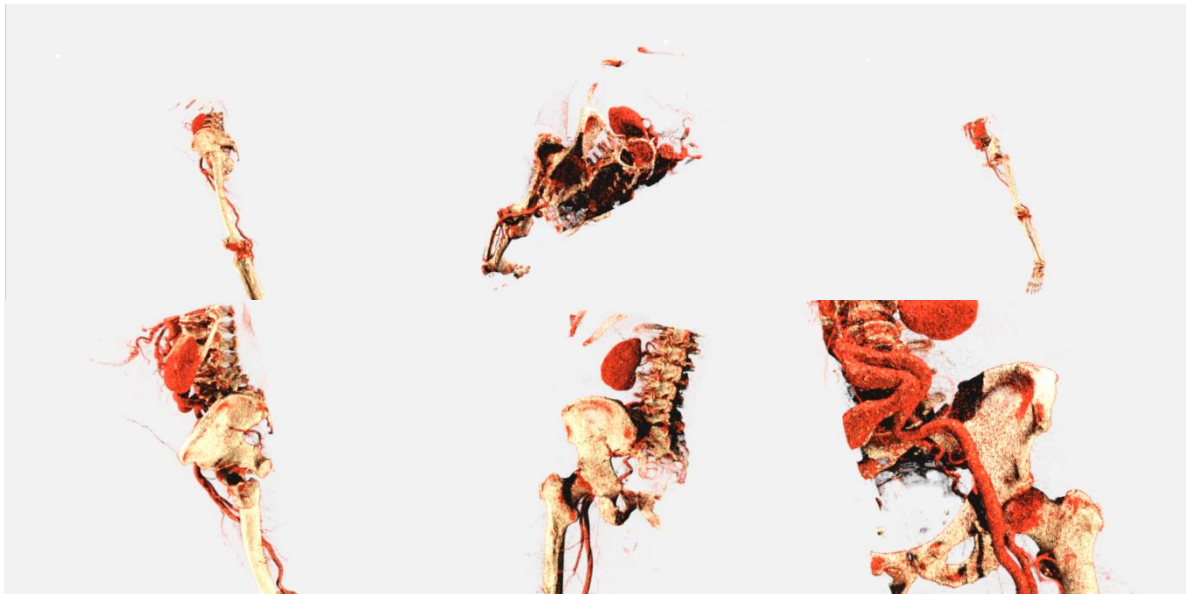
and implemented by W. Robertson<sup>22</sup>. The interpolated curve is used as an input for our system, which changes the camera position every time step, following the interpolated curve.



**Figure 26: Interpolating cluster centroids using spline interpolation. The left part of the image is generated by using unordered points compared to the right part using order points**

The cluster centroids do make sense if you look to the actual animation, e.g. the pelvis is visualized from several angles; details and overviews, as can be seen in Figure 27. These positions have to be transformed in relative positions according to the data set in order to make these positions generally applicable. The viewpoints can serve as canonical views, which for example can be used as a starting point of an investigation of a data set.

This user study method can be repeated for most body parts in order to gain information about canonical views of all body parts. The study have to be extended to a lot of participants to deduce more accurate statistical information.



**Figure 27: Six viewpoints clusters for the pelvis, extracted from user study data**

<sup>22</sup> <https://github.com/wspr/splines-matlab>



## 7 Discussion

The two different setups of the system regarding the workload distribution serves two different purposes. First, the multi-server rendering setup can be used when the output have to converge extremely fast. The system is modular, so can be extended with extra render servers to meet the given requirement of convergence time. The second setup, the multi-client error-based setup, is most likely to be used in most cases. Due to prioritizing, one render server can handle multiple concurrent users. The number of users per server is not unlimited, so in order to run the system for a vast amount of users, an appropriate number of servers have to be connected. But adding an extra server to the system does not have a major impact.

The latency of the camera interaction is not extremely low, but acceptable. A coarse measurement of the latency, whereby the server is a desktop computer i7 Nvidia with a GeForce GTX 680 graphics card and the client a desktop i5 2.8 GHz, running in a different network, results in latency values from 250 to 350 ms. A user evaluation study have to be performed to verify the hypothesis that the latency of the system is acceptable.

## 8 Conclusion and future work

In this work, an interactive remote rendering system has been presented. This system is able to render photorealistic images at interactive speed without the use of a powerful machine or device on the client side. The goal of bridging the gap between complex medical data and the patient is met by this system, since the user can easily upload and visualize the data acquired by a scan. The visualization parameters can easily be changed, e.g. camera interaction, transfer function editing, clipping, slicing, etc..

The proposed system is a perfect test-bed for user studies, which we have shown in Section 6. The information acquired in these studies is meaningful and can easily be extended due to the remote set up of the system. Therefore future work can be very promising, e.g. obtaining large quantities of data in a variety of data.

The access is free to anyone that registers for an account and anyone can upload their own data. Initially the data is for personal use, however our system promotes data sharing for scientific purposes, to this extent, data can also be easily anonymized. The application can be used for future projects and research, since the system is designed to be modular.

The system presented is a prototype version including all main features, which we required based on Section 1. There are still some aspects which can be improved or extended, which will be discussed in the following sections.

### 8.1 Stream encoder

First we could replace the FFmpeg – MPEG1 module by a more sophisticated encoder which can do a better encoding with respect to the quality and compression. CUDA could be used to encode images to an H264 stream, using hardware accelerated techniques. The client sides' webpage can handle such a stream by itself, without the need for an external library, as HTML5 browsers support h264 decoding. Probably latency and quality would be reduced significantly.

The render application generates a different series of images than video-based streams, i.e. after interaction the image contains a lot of noise, which means that it is hard to compress and only after a few iterations, the image 'smoothes' out. Based on this observation, a more clever encoding algorithm could be developed. After interaction the images are progressively increasing in quality. So a first step would be sending difference images encoded in JPEG, which make use of DCT algorithm to approach the image. After a couple of iterations, the JPEG images with their lossy compression algorithm could be switched to PNGs to encode the residual difference images. This approach seems reasonable as these images are usually sparse and can be compressed in a lossless manner without producing large files sizes.

### 8.2 Compositor

In the compositor setup, the strategy is now based on averaging the individual results of all images coming from different render nodes. An alternative approach would be to split the render canvas into an arbitrary number of sub images, e.g. four quartiles, as can be seen in Figure 28. The only thing the compositor has to do, is to combine the images into a single result. The benefit of this approach is that graphical resources are used more efficiently, i.e. each render node renders to less pixels (in this case a factor of four), which results in lower transmission load and better cache performance on the GPU.





**Figure 28: Compositor combines the output of four render nodes into one composed image**

### **8.3 Extra features**

An addition to the system would be a better collaboration framework. The different users can visualize the same data set, with the same parameter settings, but can interact with the camera freely. Now the collaboration can be more interactive, for the multiple users can investigate the data set by themselves.

Multiple presets per data set would be a beneficial feature. When the public data sets are viewed by more people, multiple presets will be stored for the same data set, e.g. one to visualize the skeleton, the other for the arteries. Eventually a crowd-generated set of presets can evolve, which could be used in future research on medical volume visualization.

## 9 Bibliography

- [1] S. Jourdain, U. Ayachit, and B. Geveci, “Paraviewweb, a web framework for 3d visualization and data processing,” in *IADIS International Conference on Web Virtual Reality and Three-Dimensional Worlds*, 2010, vol. 7.
- [2] N. Tizon, C. Moreno, M. Cernea, and M. Preda, “MPEG-4-based adaptive remote rendering for video games,” in *Proceedings of the 16th International Conference on 3D Web Technology*, 2011, pp. 45–50.
- [3] T. Hachaj, “Real time exploration and management of large medical volumetric datasets on small mobile devices—Evaluation of remote volume rendering approach,” *Int. J. Inf. Manag.*, vol. 34, no. 3, pp. 336–343, Jun. 2014.
- [4] A. Boukerche and R. W. N. Pazzi, “Remote rendering and streaming of progressive panoramas for mobile devices,” in *Proceedings of the 14th annual ACM international conference on Multimedia*, 2006, pp. 691–694.
- [5] F. Smit, R. van Liere, S. Beck, and B. Fröhlich, “An image-warping architecture for vr: Low latency versus image quality,” in *Virtual Reality Conference, 2009. VR 2009. IEEE*, 2009, pp. 27–34.
- [6] S. Shi, K. Nahrstedt, and R. Campbell, “A real-time remote rendering system for interactive mobile graphics,” *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 8, no. 3s, pp. 1–20, Sep. 2012.
- [7] C. Marion and J. Jomier, “Real-time collaborative scientific WebGL visualization with WebSocket,” in *Proceedings of the 17th International Conference on 3D Web Technology*, 2012, pp. 47–50.
- [8] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz, “Interactive visualization of volumetric data with webgl in real-time,” in *Proceedings of the 16th International Conference on 3D Web Technology*, 2011, pp. 137–146.
- [9] 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*. WSCG, 2012.
- [10] W. Hu, G.-P. Liu, and H. Zhou, “Web-Based 3-D Control Laboratory for Remote Real-Time Experimentation,” *IEEE Trans. Ind. Electron.*, vol. 60, no. 10, pp. 4673–4682, Oct. 2013.
- [11] D. Schmalstieg, “The Remote Rendering Pipeline - Managing Geometry and Bandwidth in Distributed Virtual Environments,” Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 1997.
- [12] D. Cohen-Or, “Model-based view-extrapolation for interactive VR web-systems,” in *Computer Graphics International, 1997. Proceedings*, 1997, pp. 104–112.
- [13] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel, “Scalable Remote Rendering with Depth and Motion-flow Augmented Streaming,” *Comput. Graph. Forum Proc Eurographics*, vol. 30, no. 2, 2011.
- [14] A. Wessels, M. Purvis, J. Jackson, and S. (Shawon) Rahman, “Remote Data Visualization through WebSockets,” 2011, pp. 1050–1051.
- [15] R. Smith-Bindman, D. L. Miglioretti, E. Johnson, C. Lee, H. S. Feigelson, M. Flynn, R. T. Greenlee, R. L. Kruger, M. C. Hornbrook, D. Roblin, and others, “Use of diagnostic imaging studies and associated radiation exposure for patients enrolled in large integrated health care systems, 1996-2010,” *Jama*, vol. 307, no. 22, pp. 2400–2409, 2012.
- [16] T. Kroes, F. H. Post, and C. P. Botha, “Exposure render: An interactive photo-realistic volume rendering framework,” *PloS One*, vol. 7, no. 7, p. e38586, 2012.
- [17] H. Dutagaci, C. P. Cheung, and A. Godil, “A benchmark for best view selection of 3D objects,” in *Proceedings of the ACM workshop on 3D object retrieval*, 2010, pp. 45–50.
- [18] V. Blanz, M. J. Tarr, H. H. Bülthoff, and T. Vetter, “What object attributes determine canonical views?,” *Percept.-Lond.*, vol. 28, no. 5, pp. 575–600, 1999.
- [19] P. Rousseeuw, “Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis,” *J Comput Appl Math*, vol. 20, no. 1, pp. 53–65, Nov. 1987.
- [20] J. D. Hobby, “Smooth, easy to compute interpolating splines,” *Discrete Comput. Geom.*, vol. 1, no. 1, pp. 123–140, 1986.