VolCam

Context-Aware Intuitive Touchless Interaction For Medical Volume Data

by

Rustam Alashrafov

in partial fulfillment of the requirements for the degree of

Master of Science
in Digital Media Technology

at the Delft University of Technology,
to be defended publicly on Tuesday August 22, 2017 at 1:00 PM.

Supervisors:	Dr. Anna Vilanova Prof. Dr. Elmar Eisemann
Thesis committee:	Prof. Dr. Elmar Eisemann , TUDelft Dr. Anna Vilanova, TUDelft Dr. ir. Willem-Paul Brinkman, TUDelft Dr. Ioannis Katramados, COSMONiO

TU Delft Delft University of Technology

VolCam: Context-Aware Intuitive Touchless Interaction For Medical Volume Data

Rustam Alashrafov
TUDelft
rustam.alashrafov@gmail.com

ABSTRACT

Touchless interaction has recently gained considerable attention by researchers as well as industry. Different domains are interested in implementing this technology in their solutions. Medical visualization has a special interest in this technology due to the sterile conditions in operating rooms. Exploration and detailed inspection of the scanned objects are among the most common interactions performed by professionals. These operations become more challenging when combined with touchless input. Context-aware methods exist, which facilitate navigation, but these methods are made for meshes and not for volume renderings. Hence the research question: Can these methods be extended to volume renderings and how well will they perform with touchless interaction metaphors? Metaphor and underlying VolCam algorithm are presented in this work. The metaphor allows users to perform exploration and inspection tasks on medical volume data using touchless input device - LeapMotion. The VolCam - an extension of the ShellCam algorithm, automatically maps the user input to distinct camera movements based on the current scene view by sampling the visible part of the volume. Interactive frame rates are achieved by performing computations on GPU. No pre-processing or specialized data structures are required which makes the technique directly applicable to wide-range of volume datasets.

CCS Concepts

•**Human-centered computing** → **Gestural input; Visualization design and evaluation methods;** *Pointing devices;*

Keywords

interfaces, touchless, intuitive, interaction, volume rendering, medical, NUI

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

1. INTRODUCTION

With advancements in scanning technologies such as computed tomography (CT) and magnetic resonance imaging (MRI), medical imaging has become very important in clinical practice. Clinicians use three or four-dimensional data obtained from different modalities to visualize internal structures slice-by-slice or in 3D rendering. to retrieve relevant information [28]. To make use of all this data, clinicians should be able to interact with the visualisation efficiently.

Volume rendering has become an important tool for three-dimensional (3D) data examination. It is used in different domains such as medical imaging, biological visualization, scientific computing. Modern volume visualization techniques provide an efficient representation of data which allows users to explore and understand the underlying topologies and shape structures more efficiently as well as to reduce cognitive load [8]. Cognitive load is referred to as the amount of information user has to keep and process in his working memory. Poorly presented information and redundant interface interactions can increase the cognitive load.

Most of the preliminary work is focused on real-time rendering algorithms. Advancement of these algorithms is further accelerated with ongoing graphics processing unit (GPU) innovation [24]. However, studies have shown that rendering performance on its own is not sufficient to help users understand volumes intuitively [12].

This thesis has been performed under COSMONiO's supervision. The company designs cutting-edge computervision and machine-learning systems that automate the process of extracting visual information from images under challenging conditions. Recently they have also started exploring the interfaces for touchless interaction with medical volume data. Their goal is to develop a system which will allow surgeons and medical staff to interact with the software while maintaining compliance with the sterile conditions in operating rooms (OR).

Robust navigation in 3D space using noisy input devices becomes a challenge. Camera navigation is an essential interaction for many 3D applications, and medical image viewing software is no exception. Researchers have been trying to categorize and to assess different virtual travel techniques for a long time. Bowman et al. [4] identified attributes, which describe the effectiveness of virtual travel techniques: speed, accuracy, spatial awareness, ease of learning, ease of use, information gathering and presence. In the later studies, Bowman et al. [3] distinguish three types of travel tasks, according to the user's goal:

1. Exploration, when the user has no particular area of

interest to investigate.

2. Search, where the user wants to get to a specific location in the scene. This location can be known (primer search) or not (naive search).

3. Maneuvering, which is the fine-tuning of the position of the camera around objects.

In medical data, the user is often presented with a specific dataset with a limited amount of space in the scene. Therefore, the most common type of travel is manoeuvring. Therefore, this thesis work is focused on context-aware manoeuvring techniques. Several of such techniques are presented in the following subsection.

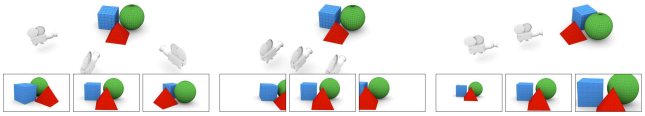

Figure 1: (Left) Rotate. (Middle) Pan. (Right) Zoom. Graphics from Jankowki and Hachet [10]

There is almost no Volume inspection happening without operations such as pan, rotation and zoom.

- Rotation refers to the orbiting motion of the camera around an arbitrary point in the view's on a virtual spherical track while the camera's focus remains on the same point throughout the operation (Fig. 1 (a)).
- Pan refers to the translation of the camera along the up and right vectors of the camera (Fig. 1 (b)).
- Zoom refers to the translation of the camera along the forward vector (Fig. 1 (c)).

However, sometimes a combination of these operations might be not so easy to perform. For example, when an object is located far away from the camera, users often find themselves lost in virtual space after several panning and rotation operations. This happens because the object may be translated outside of the view of the camera. To overcome this issue, there are different solutions, such as ShellCam [2], proposed. These techniques will be reviewed in the next section. However, most of them are developed for virtual surfaces and not volumes.

Volume data, compared to meshes, does not have well-defined surfaces. Fig. 2 demonstrates how a typical scene of the human thorax composed of meshes on the left and a typical volume rendering image on the right differ from one another. The right image consists of semi-transparent objects such as skin and kidneys. Furthermore, there are no well-defined normals. Normals are usually approximated by the gradient of the data.

Therefore, the goal of this thesis work is to investigate whether context-aware navigation methods can be extended to volume renderings and how well will they perform with touchless interaction methods. Additionally, the method has to apply to wide-range of volume datasets and should interrupt the current data set preparation pipeline as little as possible.

This thesis is structured to briefly introduce the state-of-the-art research in context-aware camera navigation which mainly focuses on derivatives of HoverCam [11]. For a better understanding of touchless-interaction and its limitations, necessary insight into the hand tracking technologies available today is provided. The innovative interaction metaphor and the underlying algorithm - VolCam are then introduced. The technique potentially assists navigation through the volume by saving time that is usually required to find the desired view and by reducing the cognitive load of the user performing the task. Implementation details, which include software and hardware specifications have been included as well to aid reproducibility of the work. Results obtained from the performed user studies are discussed. Finally, this document concludes and gives an outlook on future work.

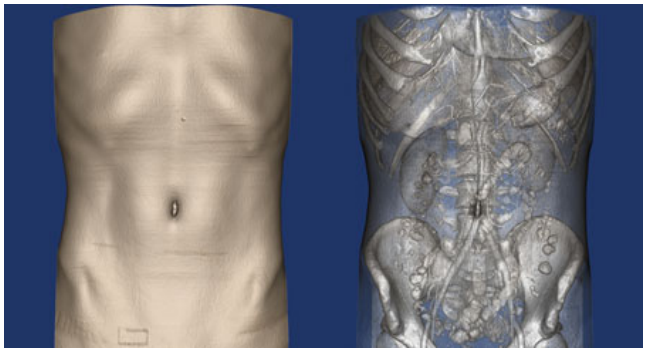

Figure 2: (Left) Rasterization. (Right) Direct Volume Rendering. Graphic from [25]

2. RELATED WORK

This section provides a quick overview of research which has been done in camera navigation techniques and their evaluation criteria.

2.1 Hover Cam

Hovercam by A.Khan et al. [11] is a base technique which tries to solve the problem of camera navigation in virtual environments (VE). It has been referenced and improved by many researchers. Authors noticed the intuitiveness of hovering around the sphere and wanted to achieve similar smoothness of motion for other objects (Fig. 3). HoverCam is looking into the closest point C on the object after the panning operation is performed. Then the look-at position is set to this point. The camera orientation is then established using the object's normal field vector and general up vector. After the translation has happened, the distance to the object is adjusted to be the same as the initial distance. Finding the closest point for the entire polygon mesh is computationally expensive and therefore not applicable for the interactive application. The authors are using a hierarchical sphere-tree structure to speedup the search process. However, the method has difficulties handling cavities (convex and concave areas). Furthermore, the proposed method is only able to navigate around the single object.

Authors of SHOCam [22] proposed to extend the HoverCam to eliminate the shakiness of the camera in areas of concavities and to add the ability to orbit around a group of objects. They have proposed to use HoverCam iteratively. At each camera position, five possible camera location are pre-computed. To achieve the smooth motion the path is interpolated between these precomputed points. This allows the camera to move at the expected distance around con-

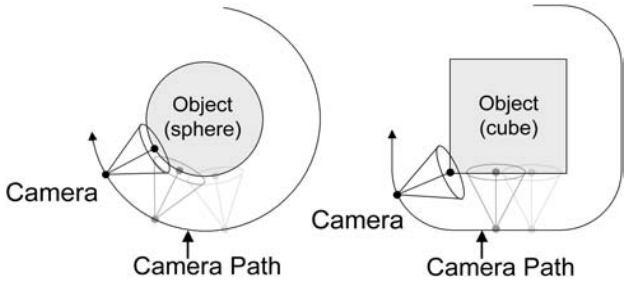

Figure 3: HoverCam Motion around Sphere (left) and Cube (right). Graphic by Khan et. al. [11]

cavities. Furthermore, they have extended the method to work in scenes with multiple objects. The authors, allow the camera to continue orbiting the object of interest without it passing through the walls of objects surrounding it. The authors use a bounding volume hierarchy for the whole scene to automatically determine the group of objects to orbit around. This criterion adapts itself to the distance of the viewer.

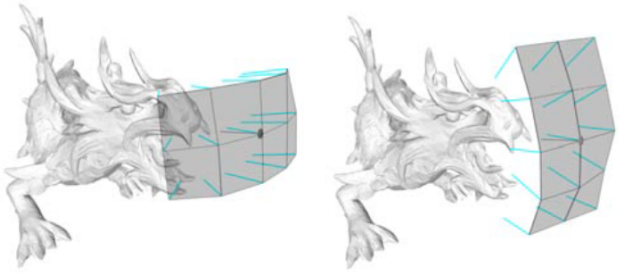

Figure 4: ShoCam: Possible camera trajectory surfaces. Graphic by Ortega et.al. [22]

To deal with geometric irregularities IsoCam's object-aware interactive camera [17] proposes to use distance fields. These distance fields are obtained from the multiscale resolution approximations of the current view-dependent approximation of the model. Iso-surfaces are precomputed in advance for different levels and further used interactively. This solution solves the issues with surface irregularities naturally since, when the camera is further away from the object, higher scale (e.g. smoother) approximations of the surfaces are used for nearest neighbour search. Another method proposes to build Curvature-Bounded Surfaces around the models[15]. The surfaces are similar to iso-surfaces used in IsoCam algorithm. They can also be computed for different distances from the object allowing the scaling effect.

The techniques mentioned above work quite well for most of the shapes and camera movement. However, they are computationally expensive, require special data structures and access to the scene's geometry. Therefore, the next method is of particular interest for interactive applications.

Diepenbrock et.al. have applied context-aware camera navigation to volume data [6]. The authors stated a decreased productivity with the direct implementation of HoverCam. Therefore, they modified the navigation metaphor to predict which movement the user wants to accomplish depending on the input and the current scene. The authors distinguish between three different types of movements (Fig. 5).

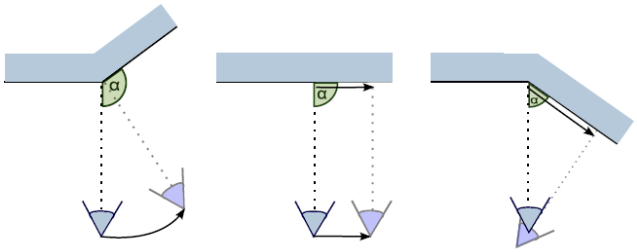

Figure 5: (a) Rotation (b) Strafing (c) Panning Graphic by Diepenbrock et.al. [6]

- Rotation - is a rotation of camera in a spherical orbit around the center point of an object, where the camera stays focused on that point.
- Strafing - is a translation of the camera along the surface of the object.
- Panning - rotation around the center of the camera itself.

Authors sample the scene depth using the first-hit approach starting from the camera position. Based on the depth information, the curvature and distance to the object are approximated. Depending on the angle of the surface, the algorithm decides whether to perform rotation, panning or interpolation between the two. Strafing is implemented by alternatively applying rotation and panning, then restoring the distance to the original distance. Furthermore, authors state the importance of collision avoidance since it is natural to move the point of view inside the volume in volumetric data. Authors' solution is to render 360-degree view around the camera and check the collision in all directions.

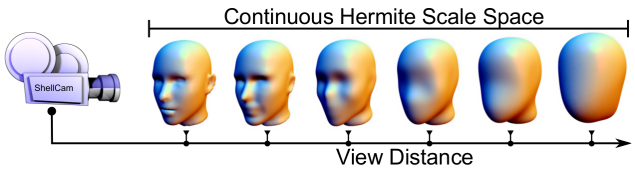

Figure 6: ShellCam: Scale-Space representation. Graphic by Boubekeur [2]

Shellcam is another technique motivated by HoverCam that looks into the distance field, just like IsoCam [2]. However, compared to HoverCam it allows for arbitrary geometry, and the complexity of the model does not play any role in the performance. The procedure also avoids shaky movement of the camera where the geometry's curvature varies quickly. Furthermore, the process eliminates the search of the procedure used in HoverCam and does not require any pre-computation and integrates well with standard graphics pipeline. ShellCam computes the weighted centroid and normal at the camera center at each iteration. This information can be naturally obtained form the Z-Buffer and Normal buffer, which makes this approach very fast.

2.2 Discussion

HoverCam [11] is a promising technique, that allows for intuitive hovering around the scene. Several works whose goal is to propose a metaphor which improves over HoverCam have been reviewed. Each of these techniques introduced interesting approaches to solving problems of the HoverCam's basic algorithm and allow for smoother interaction. ShellCam [2] is especially interesting since the algorithm proposed by the authors allows calculating the next camera position on the fly. This is achieved without the need for special data-structures by eliminating the search procedure. Furthermore, its natural extensibility to GPU makes it especially attractive for real-time applications. However, most of the techniques described in this paper have been applied to meshes, whereas Diepenbrock et.al.[6] applied a first-hit methodology which approximates meshes. This makes the technique less versatile and makes it inapplicable directly to volume renderings. For the reasons mentioned above, ShellCam became a preferred base technique for our new interaction metaphor. The original method had to be extended since volume data does not have well-defined surfaces and normals. Furthermore, volumes often consist of semi-transparent surfaces for which the behaviour of the ShellCam is undefined.

3. TECHNOLOGY

This chapter briefly introduces technologies that are used for gesture recognition. This should serve as a starting point for the readers who would like to use the main VolCam method with a different input device. Devices are compared and their limitations mentioned. However, the method described in this work can operate with any device which is capable of providing 2D movement direction and speed. This section is merely for the readers' reference that can serve as a starting point in deciding which technology to use.

As mentioned in the introduction, recent developments have brought various sensors to the market which can be used as interfaces for interaction. This section will give a short overview of sensors mentioned throughout this survey.

Color cameras (RGB sensors) are one of the most common technologies used in recognition and tracking tasks. Their popularity is partially explained by their availability. Depth information is required to recognise gestures in 3D space. RGB cameras do not provide this information by default. However, depth can be estimated from two cameras using their intrinsic and extrinsic parameters such as disparity [20]. The effectiveness of RGB sensors depends on light conditions and image resolution.

In recent years depth sensors became more consumer available and devices such as Kinect and Leap Motion have been dominating the market since. Kinect is a device by Microsoft which consists of an RGB sensor, an infrared (IR) projector, an IR sensor and an array of microphones[29]. Kinect was designed to track the human body. Software which is coming with the device allows tracking of the human skeleton. However, hackers and researches came up with solutions to recognize hand gestures as well.

The more-recently released Leap motion has a similar functionality as Kinect, but it was designed specifically to track hand gestures and fingers positions. It consists of two IR sensors and three IR LEDs [7]. Hence it can be categorised under Stereo Vision tracking systems. Leap Motion achieves sub-millimetre accuracy, which was not possible with devices in the same price range, e.g. Kinect. However, Leap Motion also has its limitations. These limitations have been described and addressed by Giulio Marin et al. [16]. Nevertheless, due to its robustness, low-cost and wide availability, Leap Motion was chosen as the hardware component of this thesis work.

4. VOLCAM

This section explains in detail the method that was used to achieve context-aware interaction with medical volume data. VolCam similar to ShellCam by Boubekeur [2] tries to follow the scene geometry in the camera's view. VolCam uses a similar methodology as ShellCam to achieve this goal. Therefore, we first introduce the ShellCam paper and later we demonstrate its extension to volume data.

4.1 ShellCam

The idea of ShellCam is to place the camera in a subspace of the object, which is in the camera's current view. The camera position is changed in a smooth, not sensitive to high-frequency variation in data, manner. This is achieved by taking into account the scale at which the scene is rendered, as illustrated in Fig. 7. The method requires only displacement movement direction and speed as input. Upon receiving an input, the camera performs the movement around the object, while taking its scale into account.

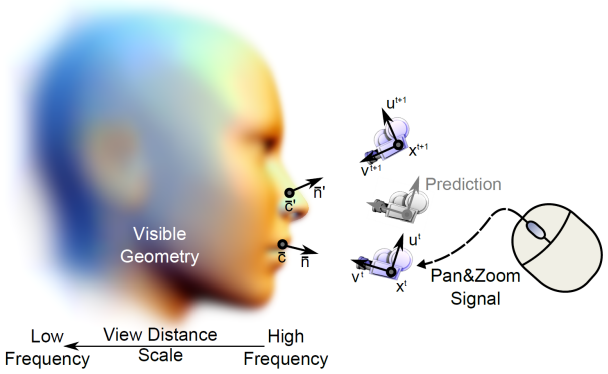

Figure 7: ShellCam Adjustment [2]

In order to explain the mathematics behind the method, variables have to be introduced first. To define the camera position and the orientation at any moment in time, a point and two orthogonal and normalized vectors are required:

$$C^t = \{\mathbf{p}_c^t, \mathbf{v}^t, \mathbf{u}^t, \mathbf{r}^t\}$$

where $\mathbf{p}_c^t \in \mathbb{R}^3$ is a camera center, $\mathbf{v}^t \in \mathbb{R}^3$ is the view direction vector, $\mathbf{u}^t \in \mathbb{R}^3$ is the up vector at time t and $\mathbf{r}^t \in \mathbb{R}^3$ is the right vector at time t, which can be derived from $\mathbf{v}^t$ and $\mathbf{u}^t$. The input will be represented by $K^t = \{\mathbf{d}^t, \iota^t\}$, where $\mathbf{d}^t \in \mathbb{R}^2$ is a two-dimensional pan input (d_x^t - displacement along x-axis, d_y^t - displacement along y-axis) and $\iota^t \in \mathbb{R}$ is a zoom input. Assume the scene is formed by objects made of polygonal meshes. Then the set of all vertices that form these meshes is:

$$P = \{\{\mathbf{p}_0, \mathbf{n}_0\}, ..., \{\mathbf{p}_m, \mathbf{n}_m\}\}$$

where $\mathbf{p}_i \in \mathbb{R}^3$ is the position and $\mathbf{n}_i \in \mathbb{R}^3$ is the normalized normal of sample i. Fig.8 Illustrates the simple scene setup.

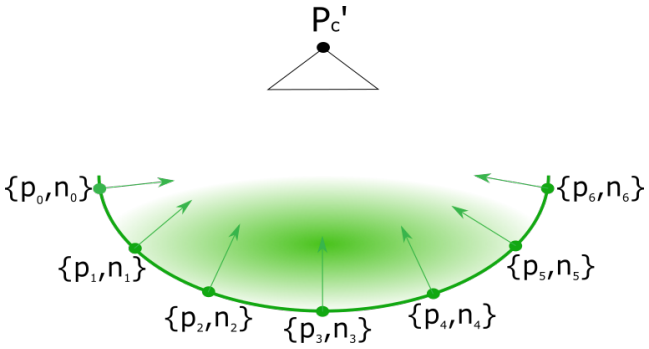

Figure 8: Sampling Demo

Green shape is a geometry with a total of m vertices. Each green dot is a sample i at position $\mathbf{p}_i$ with normal $\mathbf{n}_i$.

On Fig. 7 steps performed by the ShellCam are illustrated. Next we provide step-by-step description:

1. **Initial:** Camera is initially located at $\mathbf{p}_c^t$ with $\mathbf{v}^t = -\bar{\mathbf{n}}^t$, some $\mathbf{u}^t$ and $\mathbf{r}^t = \mathbf{v}^t \times \mathbf{u}^t$, where $\bar{\mathbf{n}}^t$ is the normal at weighted centroid position $\bar{\mathbf{c}}^t$

2. **Camera Motion Prediction:** The input K^t is received and camera is translated to $\mathbf{p}'_c$ using equation:

$$\mathbf{p}'_c = \mathbf{p}_c^t + d_x \cdot \mathbf{r}^t + d_y \cdot \mathbf{u}^t \tag{1}$$

   The rest of the camera parameters remain unmodified: $\mathbf{v}' = \mathbf{v}^t$, $\mathbf{u}' = \mathbf{u}^t$ and $\mathbf{r}' = \mathbf{r}^t$

3. **Scene Parameters Estimation** From position $\mathbf{p}'_c$ scene geometry is sampled and view dependent $\bar{\mathbf{c}}'$ and $\bar{\mathbf{n}}'$ are calculated (Equations 2, 3). Two new variables are introduced: weighted centroid $\bar{\mathbf{c}} \in \mathbb{R}^3$ and weighted mean normal $\bar{\mathbf{n}} \in \mathbb{R}^3$.These variables describe the scene's mean projection and normal. They are computed according to formula 2 and 3:

$$\bar{\mathbf{c}}' = \frac{\sum_i w^{\mathbf{v}'}_{\mathbf{p}'_c}(\mathbf{p}_i)\Pi^{\mathbf{n}_i}_{\mathbf{p}_i}(\mathbf{p}'_c)}{\sum_i w^{\mathbf{v}'}_{\mathbf{p}'_c}(\mathbf{p}_i)} \tag{2}$$

$$\bar{\mathbf{n}}' = \frac{\sum_i w^{\mathbf{v}'}_{\mathbf{p}'_c}(\mathbf{p}_i)\mathbf{n}_i}{\left\|\sum_i w^{\mathbf{v}'}_{\mathbf{p}'_c}(\mathbf{p}_i)\mathbf{n}_i\right\|} \tag{3}$$

   Equation 2 computes the weighted average of the camera position's orthogonal projection on sample i. The plane is spanned by sample's position $\mathbf{p}_i$ and normal $\mathbf{n}_i$ and is denoted as $\Pi^{\mathbf{n}_i}_{\mathbf{p}_i}(\mathbf{p}_c^t)$. These calculations are illustrated on Fig. 9. Here red dots represent the projections $\Pi^{\mathbf{n}_i}_{\mathbf{p}_i}(\mathbf{p}_c^t)$. Black circle with red stroke and black arrow represent $\bar{\mathbf{c}}'$ and $\bar{\mathbf{n}}'$ respectively.

   This way of calculating centroid generates more stable results regardless if the area of the object in the current view is concave up or concave down. Consider Fig.10. This image illustrates that the simple averaging produces a similar result for concave up object (top). However, for the concave down object, averaging places centroid inside the object and average projections push the point further from the object's surface. Therefore, it better approximates the shape curvature.

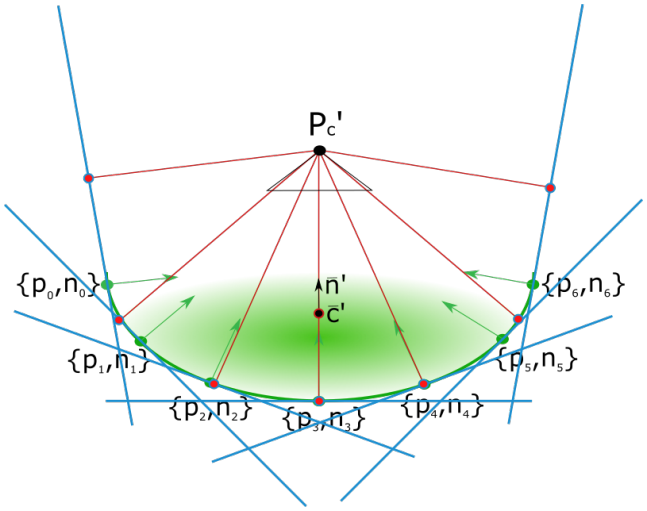

Figure 9: Scene Parameters Calculation

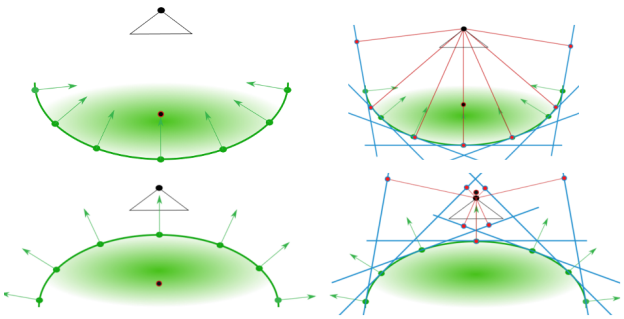

Figure 10: On the left side black dot with red boundary represents the average calculated for the given samples. On the right side same dot represents centroid calculated using projections

   As mentioned by Boubekeur [2], using projections makes the combination Hermitian, which is proven to better preserve convexity as shown by M.Alexa and A.Adamson [1]. Similarly Equation 3 is a normalized weighted sum of all sampled normals. $w^{\mathbf{v}'}_{\mathbf{p}'_c}$ is an anisotropic weight kernel centered at $\mathbf{p}'_c$. The kernel is defined as a composition of two kernels: one in screen-space, defined as the distance from the screen center, and second one defined as the distance from the sample to the camera in 3D space.

4. **Camera Motion Adjustment** The camera is set to a new position $\mathbf{p}_c^{t+1}$ according to the following equation:

$$\mathbf{p}_c^{t+1} = \bar{\mathbf{c}}' + g(\beta) * \bar{\mathbf{n}}' \tag{4}$$

   where $g(\beta)$ is a logarithmic zoom function with $\beta = \|\mathbf{p}_c - \bar{\mathbf{c}}'\|$ which captures the distance to the geometry.

The remainder of the camera parameters are defined as: $\mathbf{v}^{t+1} = -\bar{\mathbf{n}}'$. $\mathbf{u}^{t+1}$ is set to be orthogonal to $\mathbf{v}^{t+1}$ which minimizes torsion with $\mathbf{u}^t$ and $\mathbf{r}^{t+1} = \mathbf{v}^{t+1} \times \mathbf{u}^{t+1}$.

4.2 Extension to Volume Data

So far, we assumed that scenes are composed of opaque, polygonal surfaces. Despite polygonal surfaces can be obtained from volumetric data (Marching Cubes [14]), we focus on extending ShellCam to be used in direct volume rendering. In image based volume rendering samples lie on a ray i, initialized in a given pixel. The viewing ray is a ray starting at the camera center position which passes through a specific pixel on the screen i and marches through the volume as shown on Fig.11.

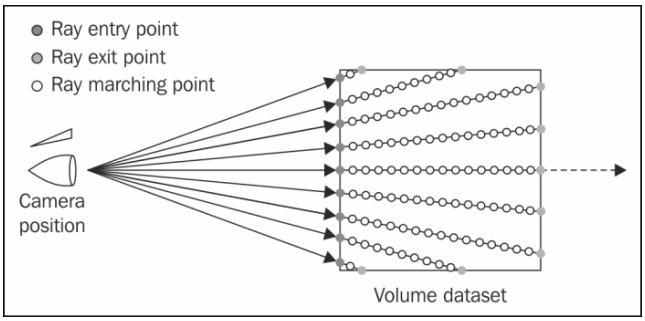

Figure 11: Viewing Rays passing through the volume. Graphic from [19]

Therefore, the set P as defined by ShellCam needs to be adapted. To start with, we deal with isosurfaces. In direct volume rendering, isosurfaces are defined by a threshold that defines a level set. This results in implicit surfaces which are not defined by their mesh. We define set of all voxels in the volume dataset as:

$$S = \{\{\mathbf{p}_0, \mathbf{n}_0, \gamma_0\}, , ..., \{\mathbf{p}_n, \mathbf{n}_n, \gamma_n\}\}$$

Where n is the total number of voxels in the dataset. Each of the voxels has a position $\mathbf{p}_i$, a normal $\mathbf{n}_i$, which is approximated by a gradient at that position and volume value (density) γ_i. To approximate the mesh, isosurfaces are extracted from the volume. With isosurfaces we can define our samples similar to the ShellCam:

$$P = \{\{\mathbf{p}_0, \mathbf{n}_0\}, ..., \{\mathbf{p}_m, \mathbf{n}_m\}\}$$

where $\mathbf{p}_i$ and $\mathbf{n}_i$ are the values of a first voxel along the ray i, which has its γ falling into the certain threshold. Gradients cannot be used directly as normals. In rasterization, users often are not interested in the back faces of objects, and they are usually culled. In direct volume rendering seeing objects from the outside as well as from inside is common. However, the gradient only gives the direction of change. Fig.12 demonstrates how normals can be arranged in the volume along the ray. Consider that we are interested in isosurface with $\gamma >= 2$. If a camera is positioned at S_3, then the normal of S_4 is pointing towards the camera as expected. However, if the camera is located on the outside (S_0), then the normal of voxel S_2 is pointing in the same direction as the viewing ray. We check $\mathbf{v}^t \cdot \mathbf{n}_i < 0$ if yes, we set $\mathbf{n}_i = -\mathbf{n}_i$. After the normals are normalized the set of samples S is ready for further processing.

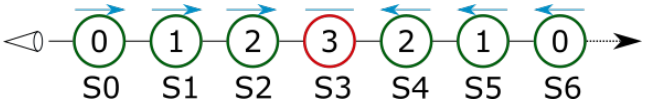

Figure 12: Demonstration of normals pointing away from the camera

Visualization of isosurfaces can be seen on Fig.13. This visualization shows the normals of the voxels that are taken into account, which is a simplified visualization of what VolCam "sees". Resulting set S can then be used directly with the ShellCam algorithm as described in Sec.4.1.

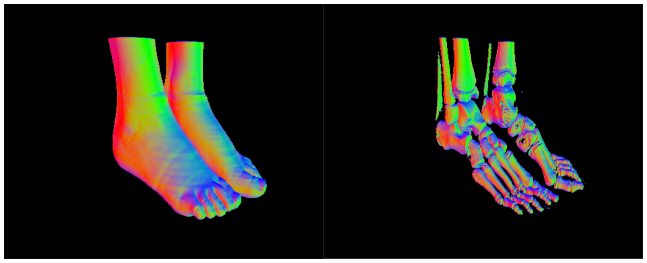

Figure 13: Left: Skin (Isovalue = 0.2) Right: Bone (Isovalue = 0.3)

A simple test setup was designed to visualize the results: For each test run, the camera starts at the same position. Panning input in the direction of the camera's **r** vector is applied for 50 iterations. This is repeated nine times. After each cycle, the camera's orientation and the rendered view is recorded. The script is designed to imitate user's pan input in the direction of the right vector of the camera. Isosurface method produced results demonstrated on Fig.14, where the blue brush line is a recorded trajectory of VolCam focused on bones whereas orange line is a trajectory of VolCam focused on the skin. It is important to notice that what the user sees and what the VolCam sees differs. While the user sees a composite view of the scene with transfer functions, VolCam sees isosurfaces.

Viewports on the sides are the camera's views from the points where the dotted arrows are pointing. We can see that trajectories of skin and bone start at the same point, but they deviate from one another over time. This solution is similar to the original ShellCam, where only the surface of an object is taken into account. However, isosurface rendering is often not preferred rendering for reasons such as it can produce false positives (spurious surfaces) or false negatives (erroneous holes in surfaces). These problems also affect VolCam robustness. Furthermore, isosurface approach does not allow for transparent surfaces. The users have to manually select which object in the volume they want to focus on. Therefore, to make the interaction more intuitive and to reduce cognitive load from users, VolCam has to follow the visualisation designed by the user using transfer functions (TF).

TF maps a single value to $RGBA$ values. Such function illustrated on Fig.15. This transfer function was used for our visualisation of the feet dataset. The horizontal axis represents the value of the volume; the vertical axis represents A assigned to the specific value. And the color of the line - color assigned to the object. Each object in the scene is

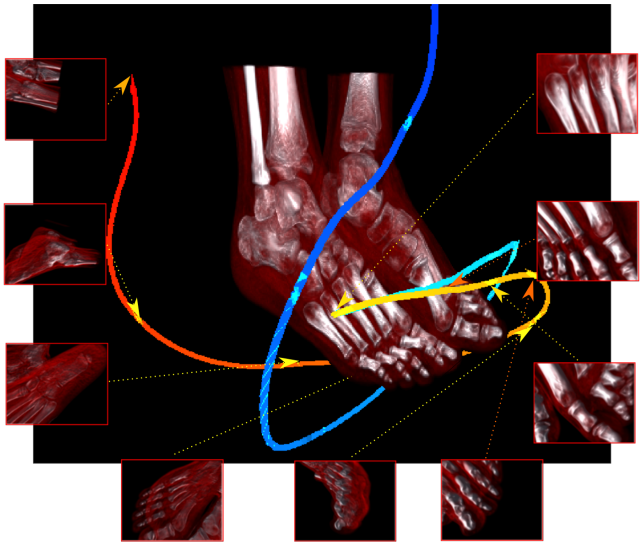

Figure 14: Trajectories produced with isosurface method. (Blue) bone isosurface. (Orange) skin isosurface. Viewports demonstrate the camera view at positions pointed by the arrows.

defined by its own transfer function. From this image, we can see that bones are given higher opacity, which means they are more prominent in the visualisation as can be seen from previous Fig.14.

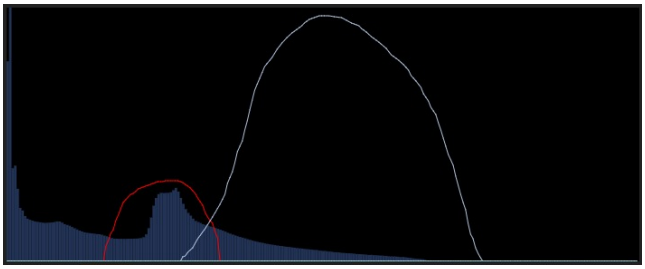

Figure 15: Transfer Functions. (Red) Skin TF. (White) Bone TF.

Similar to direct volume rendering, we can apply the idea to calculate $\mathbf{p}_i$ and $\mathbf{n}_i$. They can be computed using front-to-back compositing [18], where each sample's $\mathbf{p}_i$ and $\mathbf{n}_i$ is a composition of all projections and normals along the viewing ray i. To test if compositing works for VolCam computations compositing of separate TFs was tested. Compositing is performed by Eqn.5.

$$\begin{aligned} \mathbf{p}_i &= \sum_{j=1}^{\mu} \mathbf{p}_j \prod_{k=1}^{j-1} (1 - a_k) \\ \mathbf{n}_i &= \sum_{j=1}^{\mu} \mathbf{n}_j \prod_{k=1}^{j-1} (1 - a_k) \\ a_i &= 1 - \prod_{j=1}^{\mu} (1 - a_j) \end{aligned} \tag{5}$$

Where $a_j = f_\tau(\mathbf{p}_j)$ is an opacity given by the transfer function of object τ at voxel position $\mathbf{p}_j \in S$ and μ is the number of voxels hit by viewing ray i. The resulting VolCam view is illustrated in Fig. 16.

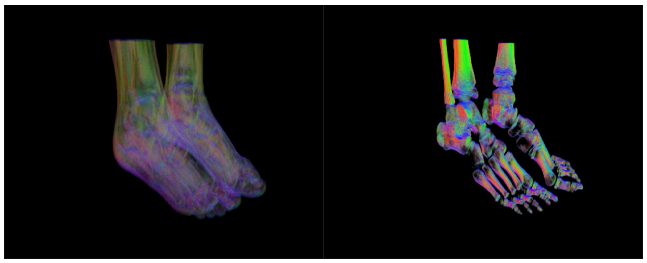

Figure 16: (Left) Skin Compositing. (Right) Bone Compositing.

By performing compositing, voxels which have higher transfer function response at that position and are closer to the camera along the ray, contribute more to the final sample P_i. This, similar to compositing of colors, makes the contributions of surfaces which are closer to the camera much higher. Hence, we get surfaces approximations without explicit extraction of surfaces. Our test with this approach produced results on Fig. 17:

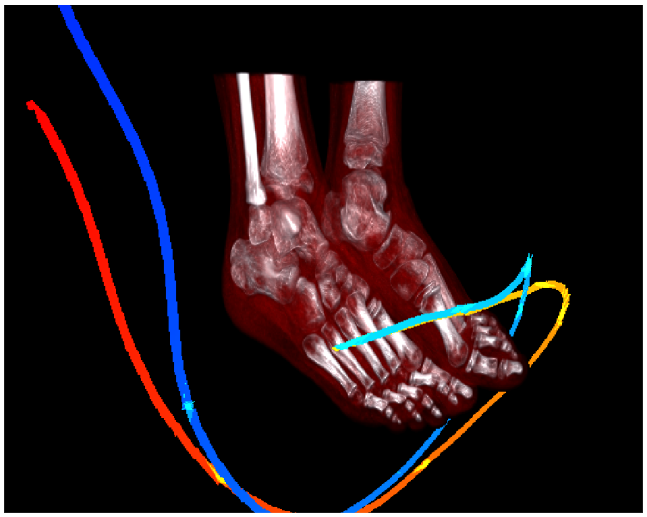

Figure 17: Trajectories produced with compositing per object method. (Blue) bone TF. (Orange) skin TF.

Once more, the blue line is the trajectory created by VolCam while focusing on the bones and orange line - while focusing on the skin. Trajectories are moving along closely. However, the blue trajectory is a bit sharper since the bone object has more details. This demonstrates that compositing can be applied to normals and meaningful results are achieved by the VolCam movement.

Therefore, all TFs (objects) in the scene viewing ray i can be composed. In this case, VolCam should see a similar image to what users see in their visualisation. The trajectory in our test was expected to be something in between the blue and orange trajectories in Fig.17. We compute a set of samples P where $\mathbf{p}_i$ and $\mathbf{n}_i$ are calculated using the same

Eqn. 5 with the difference of computing a_j:

$$a_j = \frac{\sum_{\tau}^{\phi}(f_{\tau}(\mathbf{p}_{v_j}))^3}{\sum_{\tau}^{\phi} f_{\tau}(\mathbf{p}_{v_j})}$$

where ϕ is a number of TFs (objects). This approach is imitating volume rendering. It takes into account the whole scene. VolCam in this case "sees" what is represented in the Fig. 18:

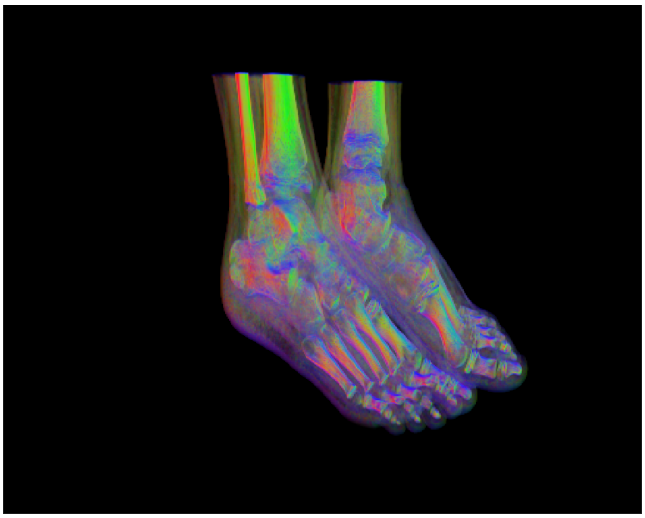

Figure 18: Compositing all objects in the volume

VolCam takes into account the whole scene in this case - skin and bones. We can see that the resulting trajectory as seen on 19 is something in between trajectories in 17. This approach lets us navigate around the whole scene without explicitly switching transfer functions to focus on or selecting isosurfaces. Compositing of different TFs (objects) gives higher weight to objects which are highlighted by the user. This reduces the amount of operations and gestures users have to perform during the navigation. VolCam is now controlled directly through visualization controls. VolCam is focused on the objects, which have TFs with higher opacity.

VolCam is very good at following the close-up curvature of objects. However, when there are several objects in the camera view, VolCam might lose its focus and switch to another surface if that surface is close to the camera or has a high opacity, hence weight. In our example with feet, imagine a situation when the camera is very close to the feet skin but is still outside of the skin. When this happens, low opacity skin voxels will receive high weight due to their small distance to the camera. This results in camera focusing on the skin rather than on the more opaque bone object. Since in medical visualisation user should get inside the object of context to investigate other objects, we can safely remove the context object from calculation during close-up inspections. Therefore, we introduced interaction system, which focuses on objects for close up inspections but uses what we call a transport medium (TM) for movement between different objects inside the volume. Transfer medium can be any object in the scene, which encapsulates the rest of the objects inside it. Usually, it is a semi-transparent object, which provides context to the rest of the scene, such as flesh or skin in medical datasets.

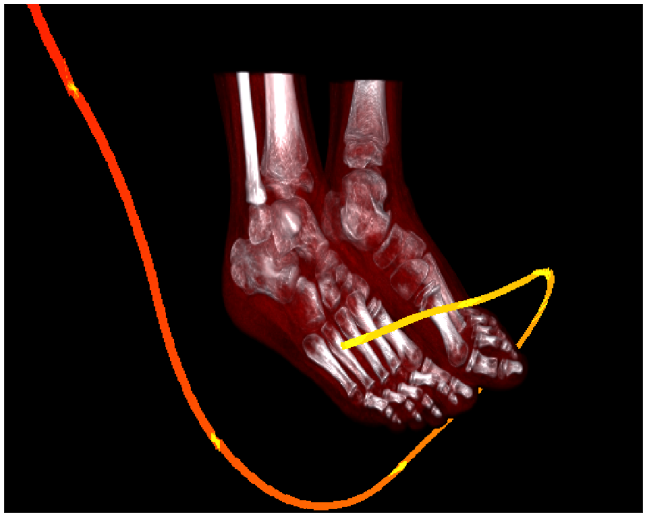

Figure 19: Trajectory produced by compositing all objects method

The user has to specify one object from the transfer function, which is going to be used as a TM. This object is then used in the VolCam calculations when the camera is far away from the scene. However, when the camera is close to the scene, TM is excluded from VolCam calculations which creates the effect of focusing on more distinct objects in the scene. We approximated the mean scene depth $\bar{z}$ by averaging the depth z_i of each viewing ray i. Depth z_i is computed using the familiar to us from 5 compositing equation:

$$\begin{aligned} \bar{z} &= \frac{\sum_i^m z_i}{m} \\ z_i &= \sum_{j=1}^{\mu} z_j \prod_{k=1}^{j-1}(1-a_k) \\ a_i &= 1 - \prod_{j=1}^{\mu}(1-a_j) \end{aligned} \tag{6}$$

Fig. 20 shows this method. While the camera is at a certain distance to the volume, context (flesh around thorax) is taken into account during the parameter calculations. When the camera is getting closer to the objects, and $\bar{z}$ is low, TM or context is not taken into account anymore. We found $\bar{z} = 0.3$ to be the best context cutoff value for both Thorax and Feet datasets. This allows the VolCam to focus on close up internal parts of the volume, in this case - aneurysm. Users see their regular visualisations and are unaware about underlying VolCam transfer function switching.

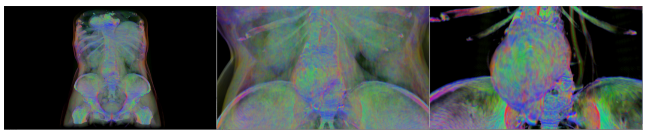

Figure 20: Effect of the distance on transfer function

5. USER INTERFACE

5.1 Interaction Metaphor

One of defining objectives of the thesis is to make the interaction touchless. A hand metaphor was chosen which is similar to touch-screen interfaces - a virtual touch-pad as described by Tuntakurn et al. [28]. This metaphor is familiar to users and, hence, does not require special training before using. Referencing the model proposed by Buxton et al. [5], we split our interaction into four distinct states: Idle, Pan, Zoom and Roll. Relation of the interaction states can be seen in Fig. 21.

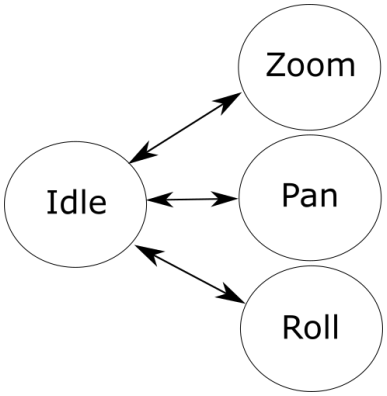

Figure 21: Interaction States

All interaction states can be instantiated only from the Idle state. This is done to simulate the touch gesture on a touch-screen. The idle gesture can be thought of as taking your hand away from the touchscreen and other gestures - as touching the screen at some initial position.

In the current implementation following gestures were chosen (Fig. 22) for interactions with system which activate particular system state :

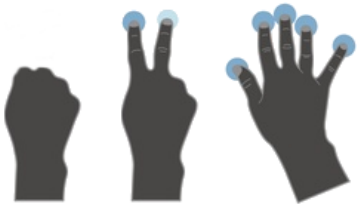

Figure 22: a. Fist gestrue b. Two Finger Gesture c. Open Palm Gesture. From [21]

- Fist - Idle
- Index&Middle fingers extended - Pan&Zoom
- Open Palm - Roll

Therefore, the same gesture can perform movement both in XY (pan) or Z (zoom) directions. When Pan&Zoom mode is activated, palm's center position is tracked. After the palm has moved minimal amount required to activate the gesture, the principal axis is derived, and camera translation along the same axis is performed by Eqn. 1, where $d^t = (\mathbf{p}_{pc}^t - \mathbf{p}_{pc}^{t-1}) * \bar{z}$ - a displacement vector between current and previous palm center positions $\mathbf{p}_{pc}^t \in \mathbb{R}^3$ projected either onto XY and $\iota^t = (\mathbf{p}_{pc}^t - \mathbf{p}_{pc}^{t-1}) * \bar{z}$ projected onto Z. It is scaled by the estimated scene depth to control the speed of movement. When the camera is close to the object, it moves slower than when it is far. We found the separation of 3D motion into two different components very important since human bodies naturally perform their movements in arcs [13]. This can result in unintentional zooming commands while panning and pan commands while zooming. After the predicted motion is performed, VolCam renders the scene and adjusts camera position and orientation as described in Sec.4.

5.2 Visualisation

To aid user interaction, visual icons are put in the right top corner of the screen. Fig.23 shows the icon usage example and all the designed icons on the right. These icons displayed to the users in the following situations:

- Hand - Correct hand is recognized by Leap Motion. The user can proceed.
- Horizontal & Vertical Arrows - User is in pan mode
- Diagonal arrows - User is in zoom mode
- Circular Arrows - User is in roll modes

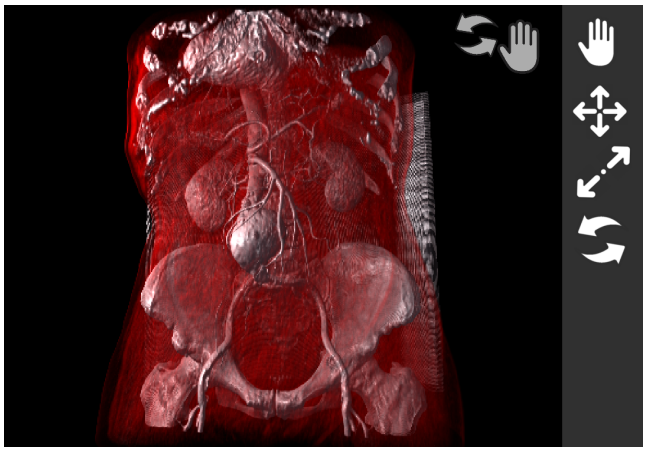

Figure 23: (Left) User Interface. (Right) Icons

6. IMPLEMENTATION DETAILS

This section describes the hardware and software architecture of the system. VolCam implementation details are provided, which allow the system to run in real-time by exploiting the graphics pipeline of modern GPU architecture.

6.1 Software

Fig. 24 demonstrates the technologies that have been used for the project. COSMONiO was using Unity for its development of the project and recommended using it for our project. To achieve easy integration of this work with parallel work that was going on at COSMONiO it was decided to continuing to work with Unity 5.5.2f1 and use *C#* as a scripting language. However, Unity is a product designed to enable easy prototyping and development of games. Unfortunately, it is not very well suited for high-performance

Figure 24: Software Architecture

applications such as medical volume rendering. It has some limitations and is a proprietary product. There is no access to the source code. Therefore, native *C++/OpenGL* Unity plugin was developed to overcome some technical challenges.

6.1.1 Data

Unity's *Texture3D* is used as data holder. Each voxel consists of four cannels where each channel is a 32bit float. R, G, B are reserved for normal vector and A reserved for voxel original volume value. The normal of a voxel is approximated by computing the gradient at a voxel S_i's position $\mathbf{p}_i$. The gradient is computed by taking componentwise differences between neighbouring voxels: : $g(\mathbf{p}_{v_i}) = \mathbf{p}_{v_{i+1}} - \mathbf{p}_{v_{i-1}}$. To make the gradient smoother, the original values are first smoothed using averaging of neighboring voxels. These computations are performed at application start. To speed up the process all volume loading operations are performed on the native *C++* plugin.

6.1.2 Main Loop Pipeline

Figure 25: Process Flow

As demonstrated in Fig. 25 as soon as the system receives input, camera's location is updated user's motion intention. From the new position, volume rendering is performed, and the necessary scene sampling is performed. In our system, sampling is done by rendering four buffers each frame Fig. 26.

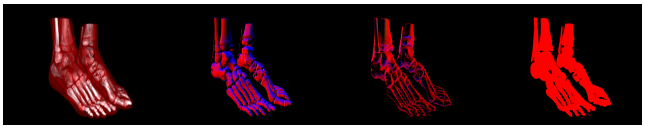

Figure 26: Buffers from left to right: Color, Normal, Projection, Depth

- *CB* - Color buffer used for visualization
- *NB* - Weighted Normal buffer
- *NB* - Weighted Projections buffer
- *ZB* - Weighted Distances buffer

Therefore, each pixel of corresponding buffer represents one sample value S_i.

6.1.3 VolCam Parameters

A fast way to calculate $\bar{\mathbf{c}}'$ and $\bar{\mathbf{n}}'$ according to Equations 2 and 3 is required. For interactive frame rates we used hardware supported MipMaps. Fig. 27 illustrates how MipMaps can be used to compute the mean value of a texture.

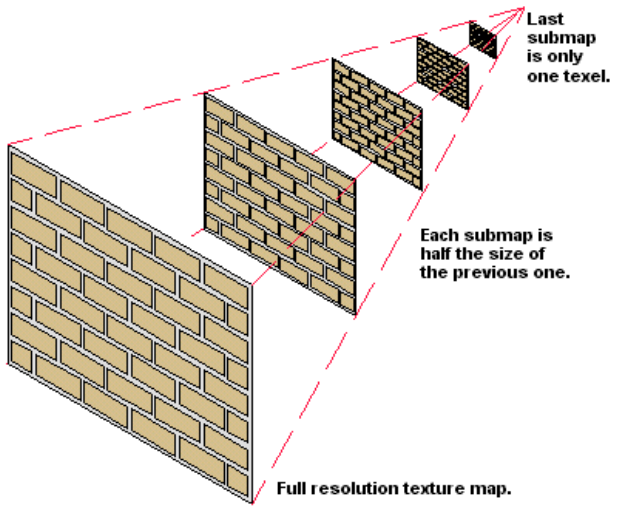

Figure 27: MipMaps. Graphic from [23]

If there are n number of pixels and h MipMap levels then the last level of the MipMap is just one pixel $\bar{\boldsymbol{\chi}}^{h-1}$ which is the average of all pixels of the initial level:

$$\boldsymbol{\chi}^{h-1} = \frac{\sum_i^n \boldsymbol{\chi}_i^0}{n} = \bar{\boldsymbol{\chi}}$$

Therefore, we can use this to quickly calculate sums and averages. The following setup was used to calculate $\bar{\mathbf{c}}'$, $\bar{\mathbf{n}}'$ and $\bar{z}'$. Corresponding buffers have to be arranged in a special way:

PB_i^0	$\{\rho_x * w, \rho_y * w, \rho_z * w, w\}$
NB_i^0	$\{n_x * w, n_y * w, n_z * w, 0\}$
ZB_i^0	$\{z, 0, 0, 1\}$

where $\boldsymbol{\rho}_x, \boldsymbol{\rho}_y, \boldsymbol{\rho}_z$ are derived projections components and $\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z$ are the normal's components of a particular sample P_i. z is the distance to the camera of the same sample i. w is the weight computed by multiplication of Gaussian kernels in 3D space and screen spaces. For the screen space kernel sigma $\sigma_s = 1$ was used. Choosing this sigma means taking into account the whole screen. However, for 3D space, no normalization is possible without second pass rendering. This happens because each new frame has new minimum and maximum distances to the voxels. Therefore, we choose $\sigma_d = 0.05$ by trial and error. Weight is computed for each voxel interactively in the shader. It is important to notice 1 in $\mathbf{ZB}_i^0$. Since there is no function in unity to quickly access number of pixels that have been modified during rendering, we use this A channel to calculate it ourselves: $\bar{\mathbf{ZB}}_a = \frac{\sum_i^n 1}{n}$.

Using these notation we can compute:

$$\begin{aligned} \bar{\mathbf{c}}' &= \frac{\bar{\mathbf{PB}}_{rgb}}{\bar{\mathbf{PB}}_a} \\ \bar{\mathbf{n}}' &= \left\| \bar{\mathbf{NB}}_{rgb} * n \right\| \\ \bar{z}' &= \frac{\bar{\mathbf{ZB}}_r}{\bar{\mathbf{ZB}}_a} \end{aligned} \tag{7}$$

Here $\bar{\mathbf{PB}}_{rgb}, \bar{\mathbf{PB}}_a$ represent RGB and A channels of the last $(h-1)$ MipMap level of the projection buffer. $\bar{\mathbf{NB}}_{rgb}$ represent RGB channel of the last MipMap level of the normal buffer. $\bar{\mathbf{ZB}}_r, \bar{\mathbf{ZB}}_a$ represent R and A channels of the last MipMap level of the depth buffer respectively.

7. EVALUATION

7.1 Participants

Seven right-handed subjects (6 male, 1 female) participated in the study as volunteers. Their ages ranged from 22 to 42, with an average age of 27. 4 of them rated their experience with navigation in 3D software as poor, 2 average and 1 proficient. 6 of test subject have never used touchless interface before, and 1 has used a Kinect to play a game once.

7.2 Setup

In our tests, a regular personal computer (PC) setup with a keyboard, a mouse, a screen and a Leap Motion connected to it through a USB3.0 port were used. PC specifications were as following:

CPU	i7 6700k
GPU	GeForce GTX1070
RAM	16 GB

With this setup and render buffers resolution of 512x512, average frame-rate was 63 frames-per-second (FPS). However, it was varying a lot between 30 and 120 FPS due to volume rendering depending on how many active pixels are in the current view and whether early ray termination is possible.

Leap motion was positioned on the table under users' hands, and the screen was placed on the eye level of users as illustrated in Fig. 28.

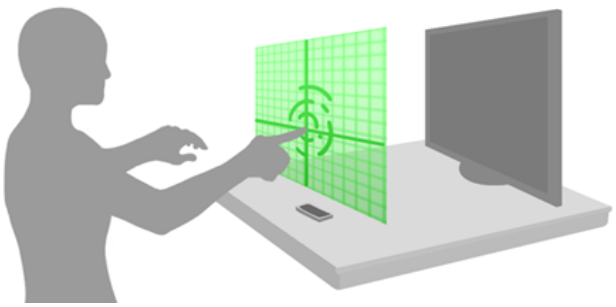

Figure 28: Hardware Setup. Graphic from [9]

7.3 Experiment

The experiment was designed to compare the method proposed in this paper with widely used virtual TrackBall (TB) metaphor. The method presented in this paper is a combination of **touchless** gesture interactions with VolCam algorithm. Therefore, from now on we will refer to this method as TVC. TB metaphor was setup in the similar to TVC way, described in section Sec.5.1, same gestures were involved as in Fig.22. The following gesture-to-command mapping was used:

- Fist - Idle
- Index&Middle fingers extended - Pan&Zoom
- Open Palm - Rotation

Rotations are performed by opening palm and translating it in the horizontal plane. Rolling is done in the in the same matter as described earlier. This metaphor was chosen as a simple metaphor which imitates interaction with pointing devices device such as mouse and very similar to TVC. This is done to compare VolCam more directly and decrease the effect created by different metaphors. We designed an experiment, in which we ask the user to perform manoeuvring tasks to evaluate the usability of our approach and compare user performance and satisfaction by using these two different techniques. About 50% of the participants in each group used the TB first, 50% the TVC first. The experiments were randomized to avoid learning effect. The task of the experiment was to manipulate the camera starting from the initial position to match three distinct target views in a specific order Fig.30. Two datasets were used: Feet and Thorax 29. Each dataset's paths were designed in such a way, that all

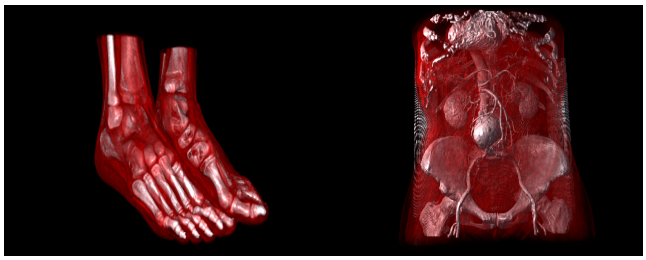

Figure 29: Datasets. (Left) Feet. (Right) Thorax.

of the three interactions had to be used: panning, zooming and rotation. First target view was positioned on the part of the dataset which is distant from the next two. This way users were forced to zoom out and navigate to the second view (object). The third view was close to the second one and contextually was the other side of the object shown in the view 2. Therefore, the transition from view 1 to view 2 is testing the movement from one object to another. The transition from 2 to 3 is testing close up manoeuvring around a specific object. Before starting the task, a brief introduction of the required hand gestures was given to the users. They had 3 - 7 minutes of practice in total until they felt they are ready to start. To increase spatial awareness of the user inside the volume, demonstration, explaining the objects represented in the target views and where they are located inside the volume, was given. Users had the current target view in front of them, next to the screen at all times. Users had to try to get the view as close as possible to the target view. When they were satisfied with the current view, time, the camera position and orientation were recorded and users were asked to proceed to the next target.

At the end of the experiment, participants were asked to fill in the form, where they had to rate their experience, answer questions which metaphor they prefer. Furthermore, the form asked to comment on what features they missed if any, and for some additional commentaries.

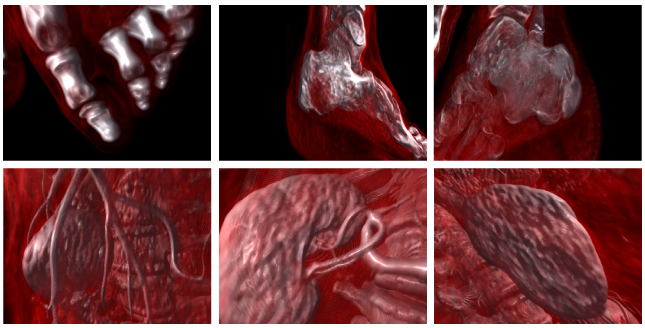

Figure 30: Target Views in order from left to right. (TOP) Feet. (Bottom) Thorax.

7.4 Experiment Results and Observations

Users on average spent more time performing tasks using TVC metaphor. Users had difficulties predicting where the VolCam would go next. However, four users finished the transition from view 2 to view 3 faster using TVC. This shows that our assumption that VolCam can perform better for close up inspections is confirmed. In general, users preferred standard Trackball metaphor. From the participants' feedback, we learned that TrackBall feels intuitive since they are already familiar with it. It was hard for them to predict how the VolCam will move. They admit they could not grasp the concept of following the surface. They mentioned in the comments they found themselves planning the route from one object to another one, which increased cognitive load. Unfortunately, during the experiments, the Leap Motion device was not performing at its best due to lighting conditions and would often lose track of the hand and often recognize right hand as left one. This would stop the interaction and users had to reset their hand gesture. This decreased the overall experience. Three of the users commented that they would prefer to use mouse and keyboard which would not distract them from the experience of VolCam.

From our observations, it was obvious that the time given to the users was not sufficient to get used to the TVC metaphor. While working with TVC metaphor, instead of moving the surface and directly get to the other side of the object, users would often zoom out and try to position the object in the center of the screen. This work-flow is very similar to the traditional TrackBall metaphor and is not efficient while using VolCam. We have also noticed that unpredicted motion of the camera would often occur due to the noise in the dataset, especially in the Thorax dataset. In some specific camera views, opaque voxels appear close to the camera which has very high weight. This weight makes VolCam focus on them and thus makes sudden, unpredicted motion. Problem is illustrated on Fig.31

The place users were struggling have been recreated and rendered in debug mode. The green highlight indicates the objects the VolCam considers to be of importance. When performing transition 1 to 2 in Thorax dataset, it goes smoothly

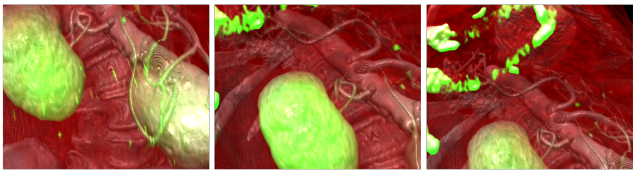

Figure 31: Noise Visualisation

until the high opacity voxels appear in the left corner of the screen and additional input signal in that direction makes the camera to quickly change its focus to those voxels. Since the noisy data does not look like a surface in the visualisation itself (Fig.32) , users would not recognize it as a potential surface for VolCam to switch. This created additional misunderstanding. Therefore, addressing this problem would

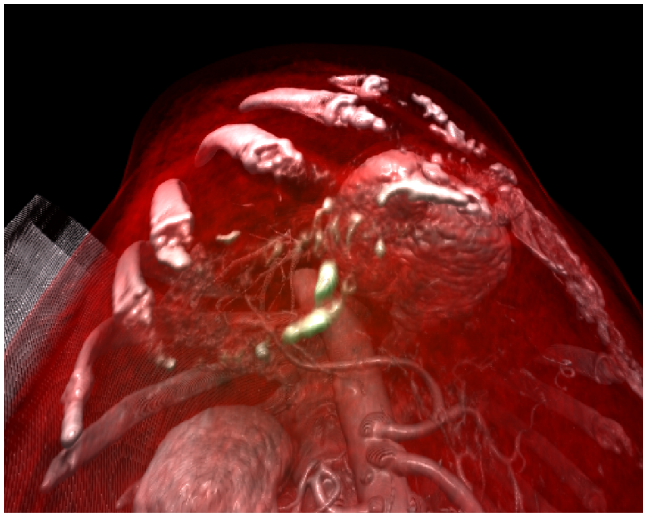

Figure 32: Opaque bone pieces

drastically increase the performance of VolCam.

8. RESULTS

VolCam allows to follow objects on volume datasets defined by their TFs. The method works the best for close up inspection of objects and with objects that ahve smooth surfaces as illustrated in 19. Furthremore VolCam behaviour is illustrated in Fig.33, where the camera is moving along the surface of aneurysm in the Thorax dataset.

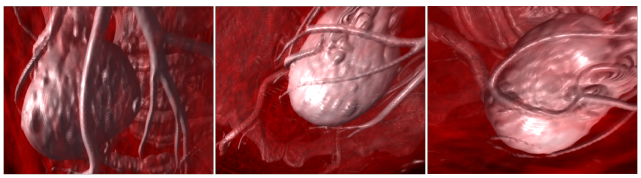

Figure 33: Sequence of VolCam views in order from left to right

However there are several points of improvements to the algorithms which will be discussed in the remainder of the section.

- **Noise**: As illustrated in Fig.31 noise creates problems for the VolCam algorithm. Therefore, the volume should be filtered. Opening - morphological operator can be applied to the volume to reduce the amount of noise.

- **Occlusion**: Another problem encountered while working with VolCam in crowded spaces is the fact that, new objects can appear between the object of interest and the camera, during the change of the camera view. This effect is illustrated in Fig.34. While following the surface of the kidney, suddenly the rib cage appears very close to the camera. The camera starts switching its focus to the rib case rather than continuing to follow the surface. The camera starts switching the objects instead of continuing following the surface of the kidney. In this case, to continue following the surface, users have to zoom in closer to the surface before continuing movement. This makes users plan their way through the volume which increases cognitive load. The occlusion problem is further exaggerated when the

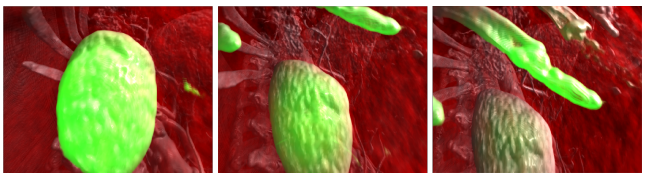

Figure 34: Occlusion problem demo

camera ends up inside an opaque object. This makes users completely blind since all they can see is voxels of that object covering the whole screen. However, VolCam continues to work by focusing on inner surfaces of that same object. This combination of blind manoeuvring takes users into unpredicted locations. We have investigated a possible solution of segmenting the data by defining separate transfer functions for each object or creating masks and let the user manually choose a region of interest. This solution works pretty well and allows keeping the focus on the object even when it is occluded. However, it creates another step in data preparation and forces manual input from the user to choose which segmented part of the volume to follow Fig.(35). More interesting research question would

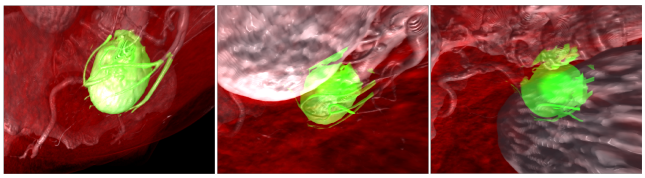

Figure 35: Segmentation solution demo

be: How to decide which object the user is following and be able to lock onto that object upon receiving the user command? This can be done by determining at what distance from the camera the object of interest is located and locking into that distance. This will make VolCam ignore objects further or closer from the camera than the object of interest. This solution would also reduce the problem of noisy inputs appearing on the screen.

- **Hand Tracking** TVC was not very stable during our tests. Tracked hands were lost at times, and LeapMotion took some time to recognize the hand again. Therefore, a more robust implementation is required. A quick fix can be to not stop the interaction when the hand is not recognized only for a couple of frames. This can be done by keeping the position of the hand in memory and projecting hands motion for a fraction of a second until the Leap Motion detects the hand again. On top of that, one could easily let active gestures, like panning, to continue even when LeapMotion lost track of right hand and detected the same hand as the left one.

- **Gestures** Our gestures were chosen to be as simple and familiar as possible to the users. This was done to analyze VolCam performance better. However, there has been research on ob NUI gesture interactions. Other gesture metaphors should be tested with the VolCam algorithm. These are a couple of promising metaphor that could be investigated further investigated [26] [27] .

9. CONCLUSION

In this work we investigated the problem of intuitive touchless interaction for medical volume datasets. Precise and robust manoeuvring around the 3D volume is further complicated when users have to perform repetitive sequences of motions such as pan, rotation to get the desired view. We have reviewed context-aware solutions which perform combinations of these movements on behalf of the user. However, most of these solutions are developed for virtual, opaque surfaces made of meshes. Therefore, this thesis work set a goal to investigate whether context-aware navigation methods can be extended to volume renderings and how well they will perform with touchless interaction methods.

VolCam - an extension of ShellCam to volume datasets, which supports gesture metaphor has been proposed and developed as a solution to the given problem. VolCam tracks the current visualisation composed by the user defined TFs. Our method does not require any data preparation and adapts to the changes in visualisation in real time. User studies showed that our method performs well for close up inspection of surfaces and is faster than the conventional trackball metaphor. However, for general volume navigation users operated faster with Track Ball. Users have also expressed that they prefer Track Ball over the Touchless VolCam metaphor for general navigation. There is room for improvement of the proposed method. The problems associated with the current solution such as noise in the volume, occlusion of objects of interest have been discussed and analyzed. The methods, which can potentially solve the problems have been proposed.

10. FUTURE WORK

There are several points of improvement. The most important once which will directly improve the VolCam are described in Sec.8. Noise in the volume should be addressed since it greatly affects the robustness of VolCam and can improve the overall experience of the users. Several possible

solutions have been proposed to deal with occlusion which interrupts the user interaction and creates unexpected VolCam movement. One potential solution to investigate is to create a user interface which enables users to control the objects they are focusing on manually. Another possible solution is to check for collisions and not allows users to go inside objects and restrict their movements. A similar method is employed by Depenbrock et.al. In their system for virtual colonoscopy [6]. Further investigation is needed to explore gesture metaphors which can better suit for the interaction with VolCam. Overall system experience can also be improved by improvements in hand tracking and rendering performance.

11. REFERENCES

[1] M. Alexa and A. Adamson. Interpolatory Point Set Surfaces - Convexity and Hermite Data. *ACM Trans. Graph.*, 28(2):20:1–20:10, May 2009.

[2] T. Boubekeur. Shellcam: Interactive geometry-aware virtual camera control. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 4003–4007. IEEE, 2014.

[3] D. Bowman, E. Kruijff, J. J. LaViola Jr, and I. Poupyrev. *3D User Interfaces: Theory and Practice, CourseSmart eTextbook*. Addison-Wesley, 2004.

[4] D. A. Bowman, D. Koller, and L. F. Hodges. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *Virtual Reality Annual International Symposium, 1997., IEEE 1997*, pages 45–52. IEEE, 1997.

[5] W. Buxton. A three-state model of graphical input. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, INTERACT '90, pages 449–456, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co.

[6] S. Diepenbrock, T. Ropinski, and K. Hinrichs. Context-aware volume navigation. In *2011 IEEE Pacific Visualization Symposium*, pages 11–18, March 2011.

[7] B. R. D. F. Frank Weichert, Daniel Bachmann. Analysis of the accuracy and robustness of the leap motion controller.

[8] L. Gallo. A study on the degrees of freedom in touchless interaction. In *SIGGRAPH Asia 2013 Technical Briefs*, SA '13, pages 28:1–28:4, New York, NY, USA, 2013. ACM.

[9] S. Hanselman. Leap motion: Amazing, revolutionary, useless, August 2013. [Online; accessed August 8, 2017].

[10] J. Jankowski and M. Hachet. A survey of interaction techniques for interactive 3d environments. In *Eurographics 2013-STAR*, 2013.

[11] A. Khan, B. Komalo, J. Stam, G. Fitzmaurice, and G. Kurtenbach. Hovercam: interactive 3d navigation for proximal object inspection. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 73–80. ACM, 2005.

[12] B. Laha, K. Sensharma, J. D. Schiffbauer, and D. A. Bowman. Effects of immersion on visual analysis of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):597–606, April 2012.

[13] M. Leap. Vr best practices guidelines. June 2015.

[14] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987.

[15] L. Malomo, P. Cignoni, and R. Scopigno. Generalized trackball for surfing over surfaces. In *STAG: Smart Tools and Apps for Graphics*. Eurographics, 2016.

[16] G. Marin, F. Dominio, and P. Zanuttigh. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 1565–1569, Oct 2014.

[17] F. Marton, M. B. Rodriguez, F. Bettio, M. Agus, A. J. Villanueva, and E. Gobbetti. Isocam: Interactive visual exploration of massive cultural heritage models on large projection setups. *Journal on Computing and Cultural Heritage (JOCCH)*, 7(2):12, 2014.

[18] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[19] M. M. Movania. Opengl development cookbook, June 2013. [Online; accessed August 8, 2017].

[20] J. Mrovlje. Distance measuring based on stereoscopic pictures.

[21] J. Mula. Leap motion: The power is on your hands, October 2015. [Online; accessed August 8, 2017].

[22] M. Ortega, W. Stuerzlinger, and D. Scheurich. Shocam: A 3d orbiting algorithm. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 119–128. ACM, 2015.

[23] PCMag. Mip mapping. [Online; accessed August 8, 2017].

[24] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[25] A. C. Sandy McKenzie, Lisa Avila and S. Jhaveri. Volume rendering improvements in vtk, 2014. [Online; accessed August 8, 2017].

[26] J. Shen, Y. Luo, X. Wang, Z. Wu, and M. Zhou. Gpu-based realtime hand gesture interaction and rendering for volume datasets using leap motion. In *2014 International Conference on Cyberworlds*, pages 85–92, Oct 2014.

[27] K. R. Sivaramakrishnan, G. K. Raja, and C. G. Kumar. A touchless interface for interventional radiology procedures. In *2015 International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*, pages 130–133, Oct 2015.

[28] A. Tuntakurn, S. S. Thongvigitmanee, V. Sa-Ing, S. S. Makhanov, and S. Hasegawa. Natural interaction on 3d medical image viewer software. In *The 5th 2012 Biomedical Engineering International Conference*, pages 1–5, Dec 2012.

[29] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, Feb 2012.