FAST CONTEXTUAL VIEW GENERATION AND REGION OF INTEREST SELECTION IN 3D MEDICAL IMAGES VIA SUPERELLIPSOID MANIPULATION, BLENDING AND CONSTRAINED REGION GROWING

by

Ken Lagos, BSc, Ryerson University, Canada, 2015

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2019

FAST CONTEXTUAL VIEW GENERATION AND REGION OF INTEREST SELECTION IN 3D MEDICAL IMAGES VIA SUPERLLIPSOID MANIPULATION, BLENDING AND CONSTRAINED REGION GROWING

by

Ken Lagos, MSc, Ryerson University, Toronto, Canada, 2019

ABSTRACT

This thesis presents a 3D widget user-interface (UI), super-ellipsoid shape primitives and a customized volume rendering algorithm that together create a system effective for exploring 3D medical images and for selecting a 3D region within these images. Using a "painting" metaphor, the widget UI supports the fast and precise positioning of a super-ellipsoid shaped paint "blob". The paint blob can be "deposited" and automatically blended with previously deposited blobs to form arbitrarily-shaped regions enclosing target image features. The rendering of these "focus" regions can be controlled separately from the surrounding contextual region, allowing medical experts to examine and measure image features relative to the context. The system's core algorithms are designed to execute on *Graphics Processing Units* (GPUs), resulting in real-time interaction and high-quality visualizations. The *focus plus context* visualization system presented in this thesis is validated via a user study and a series of experiments.

ACKNOWLEDGEMENTS

I would like to thank my friends and family for their support throughout these years in my decision to pursue a master's degree. It has been a long journey with a lot of ups and downs, but I think it was worth it in the end. I would also like to thank Dr. Tim McInerney and Dr. Kosta Derpanis for their support and guidance as I probably could not have done it without their support.

TABLE OF CONTENTS

List of Acronyms

1. ROI - Region of Interest
2. GPU - Graphics Processing Unit
3. TF - Transfer Function
4. VR - Volume Rendering
5. DVR - Direct Volume Rendering
6. GUI - Graphic User Interface
7. FPS - Frames per Second
8. MRI - Magnetic Resonance Imaging
9. CT - Computer Tomography
10. PET - Positron Emission Tomography
11. DICOM - Digital Imaging and Communications in Medicine
12. RGBA - Red Green Blue Alpha
13. LOD - Level of Detail
14. SDK - Software Development Kit
15. API - Application Programming Interface
16. ESS - Empty Space Skipping

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

The visualization of 3D medical volume images has evolved rapidly over the past two decades. The emergence of powerful Graphics Processing Units (GPUs), containing hundreds of processing cores, has enabled the development of visualization algorithms that process these data volumes in parallel. This hardware advancement has provided medical professionals with the capability of viewing, manipulating, and exploring high-resolution 3D renderings of the data in real-time. Consequently, the use of these 3D visualizations for examining and measuring specific anatomical structures or pathologies, especially when viewed in the context of neighboring structures, is now an important step in disease diagnosis, treatment planning and the planning of surgeries. Direct Volume Rendering[1] (DVR) is the standard algorithm for generating the visualizations, especially for CT[2] scans. Expert users utilize Transfer Functions (TFs) to control the visual output of a DVR algorithm. A TF maps the scalar field values - commonly referred to as intensity values - stored in a 3D medical image to optical properties of opacity[3] and color. A TF is commonly defined using a curve or a piecewise linear polynomial and a graphical user interface (GUI) enabling the user to control the curve shape (See Section 2.1). A TF GUI provides users with the ability to quickly select classes of anatomical structures, such as skin or bone or arteries, for rendering.

---

[1] In computer graphics, direct volume rendering is an algorithm that samples 3D volumetric datasets along rays emanating from the viewpoint, mapping the sample values to a color and opacity and accumulating these color and opacity values to generate a final color and opacity for an array of screen pixels.
[2] Computed Tomography
[3] In computer graphics, opacity describes the degree to which an object is opaque. That is, it controls the transparency of an object.

Despite the extensive amount of research in the field of volume rendering and transfer function (TF) design [1], quickly defining a single TF that generates the desired view of a spatially localized *focus* region within the data volume, or of specific anatomical structures, remains a non-trivial task. Specifically, selecting an individual structure, a part of a structure, or an arbitrary volumetric region of interest (ROI) containing several target structures is more difficult with these global functions. This may be due to the fact that TF specification is primarily a mapping of ranges/characteristics of voxel intensity values[4] (i.e. it is data attribute based), and thus doesn't provide the user with the ability to localize specific features or regions in the volume image without some additional control mechanisms. These additional mechanisms, such as 2D transfer functions, provide more control to the user over the visualization output but often at the cost of more complex user interfaces that are still primarily *indirect* in nature. Fast and simple *direct* selection of a spatial sub-volume can complement a TF specification process by supporting separate and more easily defined TFs for the "focus" region within the ROI, and for the context region surrounding it. In particular, the separate TF can be set to visually "cut away" occluding data within the ROI in order to reveal interior target structures, while also providing the option of maintaining a contextual view of the surrounding volume. Conversely, a dedicated TF for a ROI can also be used to select and highlight features within the ROI using some sort of distinctive rendering style in order to aid in the visual analysis. Another use of a user-defined ROI is as input to a semi-automatic segmentation[5] algorithm, in order to either constrain the algorithm or provide it with an accurate

[4] A voxel represents a scalar field value in a 3D image. It can be considered a 3D extension of a pixel. The scalar field value is commonly referred to as the *intensity* value.

[5] Segmentation is the process of labelling voxels in an image with the goal of dividing it into multiple parts. Segmentation algorithms are used to identify objects and to allow for further analysis of the objects.

initialization, especially in noisier 3D medical images such as MR[6] volumes where TF-defined volume rendering of un-segmented images is often less effective. This "constraint envelope" strategy may improve the robustness of the segmentation algorithm as well as potentially minimize the often tedious and time-consuming manual editing phase [2] [3] [4]. A user-defined ROI constraint/initialization may be especially useful in scenarios where segmentation algorithms do not perform well, such as segmenting objects in noisy images.

Designing interaction techniques enabling a user to explore a volume or to quickly generate a contextual view by directly selecting a focus ROI or selecting a specific target structure in a volume rendered image is a challenging human-computer interaction (HCI) task. An interactive ROI selection technique needs to provide the user with the ability to overcome the problems with the TF specification approach by supporting functionality for the fast and controllable selection of objects adjacent to other objects that have been mapped by the TF to similar opacity and color values. Furthermore, target ROIs may vary from regularly shaped regions, such as rectangular blocks or spheres, to arbitrarily complex, curving shaped objects/features, or to elaborate branching objects such as vasculature. In addition, the TF specification may result in a volume rendering in which target ROIs contain several visually disconnected objects or ROIs that are visually occluded by parts of other structures. It is also an important requirement of interactive ROI selection techniques to support the fast modification of the ROI as a user is exploring the volume from new viewpoints. That is, visual analysis of medical images is often an iterative process of initial view generation and visual examination, followed by view changes or refinements (via volume navigation) and additional visual analysis. Finally, the interaction issues

[6] Magnetic Resonance

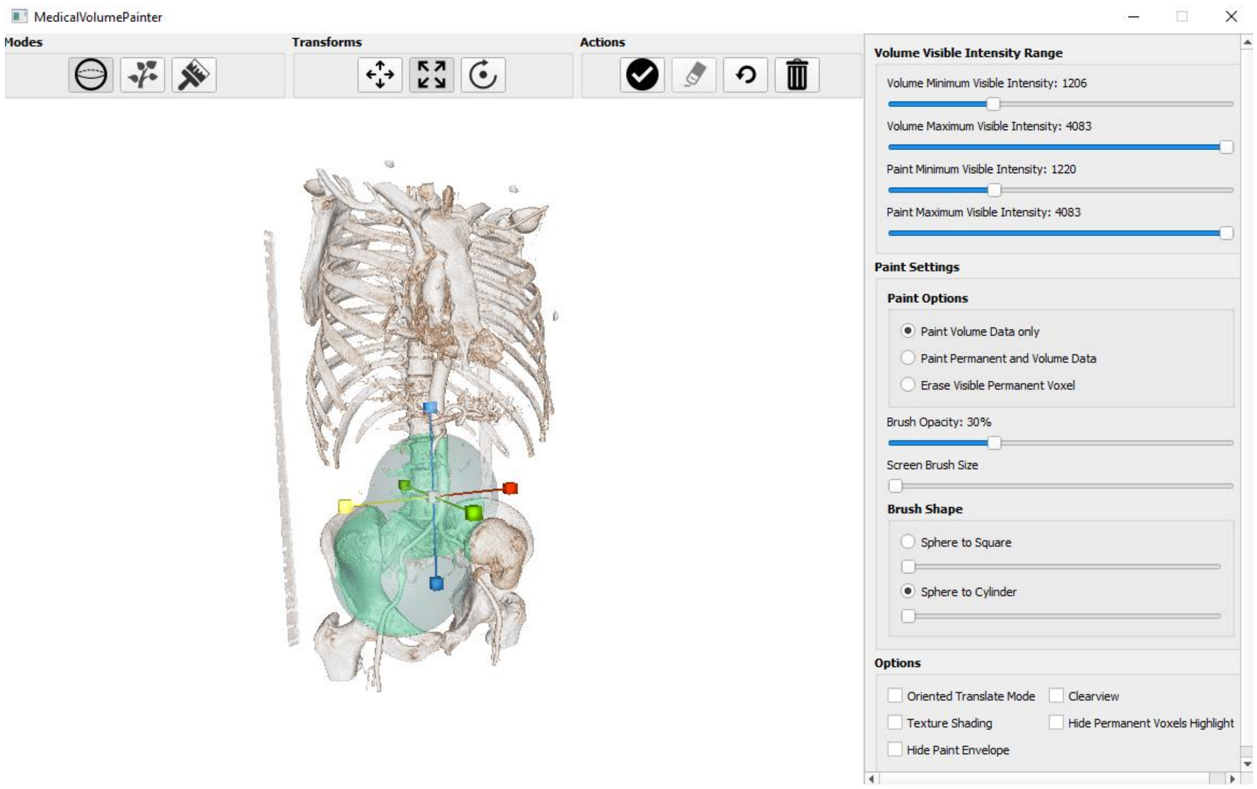
are inherently tied to issues of the user's depth perception of the target ROI as well as issues involving the design of appropriate supporting visual cues, as well as signifiers[7] to indicate affordance, that guide the user's actions during the ROI selection process. In summary, the goal is to be able to simply and efficiently explore, select, and optionally remove regions of any shape or level of occlusion, in order to examine and focus on structures and their relationship to surrounding structures.

1.1 Thesis Statement

This thesis presents an interactive view generation and ROI selection technique that is based on the manipulation and rendering of mathematically-compact and blend-able convex shape primitives known as super-ellipsoids. Specifically, a user interface (UI) design that uses 3D widgets for manipulating the super-ellipsoids is presented (Figure 1). The main hypothesis is that the combination of the 3D widget UI, blend-able super-ellipsoids blobs, and a standard TF interface is an effective paradigm for quickly creating useful views. The thesis will attempt to quantitatively evaluate the UI efficiency and accuracy, as well as qualitatively evaluate the ease of use, flexibility, and rendering control of the super-ellipsoid based technique and its associated UI for selecting and visualizing a wide range of target ROIs and anatomical structures from 3D medical images in order to generate effective contextual views. The 3D widget-based UI (referred to as a blob tool) for manipulating blobs is compared to an alternative UI based on a *surface*

[7] According to Norman [55] an "affordance is a relationship between the properties of an object and the capabilities of the agent that determines just how the object could possibly be used". Norman then defines the term signifier as "any mark or sound, any perceivable indicator that communicates appropriate behavior to a person". Put simply, Norman states: "Affordances determine what actions are possible. Signifiers communicate where the action should take place". Affordances are what an object can do (truth). Perceived affordances are what one thinks an object can do (perception). Signifiers make affordances clearer (closing the gap between truth and perception). Signifiers often reduce number of possible interpretations and/or make intended way of using an object more explicit."

paintbrush [5] and to a well-known *screen*-based paintbrush ROI selection technique. Several experiments demonstrating the use of the view generation technique for creating a wide range of contextual views are also performed in order to further validate the hypothesis.



Volume Visible Intensity Range

Volume Minimum Visible Intensity: 1206

Volume Maximum Visible Intensity: 4083

Paint Minimum Visible Intensity: 1220

Paint Maximum Visible Intensity: 4083

Paint Settings

Paint Options

[✓ Paint Volume Data only]
Paint Permanent and Volume Data
Erase Visible Permanent Voxel

Brush Opacity: 30%

Screen Brush Size

Brush Shape

Sphere to Square
[✓ Sphere to Cylinder]

Options

Oriented Translate Mode
Clearview
Texture Shading
Hide Permanent Voxels Highlight
Hide Paint Envelope

Figure 1: *A snapshot of the graphical user interface and blend-able super-ellipsoid blobs with a 3D widget interface.*

1.2 Contributions

A super-ellipsoid based interactive volume navigation and ROI selection technique allows for the dynamic generation of "focus plus context" views [6] in a volume rendered image. Super-ellipsoids, referred to as "blobs" in this thesis, are defined using implicit functions and can be seamlessly and tightly blended to form volumetric "paint" that envelopes anatomical regions of any geometry or topology in the volume image. The blobs also provide a convenient mechanism for interactively controlling and constraining a GPU region growing segmentation algorithm [7]. That is, connected sets of visible voxels belonging to a target anatomical structure are "grown" (i.e. selected and highlighted) from a user-selected "seed" voxel, in real time. A widget-based UI is used to change the position, shape, and size of the blob and thereby controls region growth within it. Furthermore, inside a selected ROI defined by the blobs, a special TF may be applied that controls color and opacity separately from the surrounding contextual region, including the capability of rendering all voxels within the ROI invisible and thereby allowing users to quickly remove occluding structures.

This thesis significantly expands on and improves upon previous work [5] that utilized a more primitive, primarily spherical paint blob, along with a blob manipulation UI that utilized a data iso-surface "painting" interaction metaphor – which is referred to in this thesis as a *surface paintbrush*. In summary, the contributions of this thesis are:

1. A 3D widget UI for positioning, orienting and resizing super-ellipsoid shape primitives in the 3D volume space. This "blob painting" UI enables blobs to be quickly manipulated from any scene viewpoint using "handles". Widget handles are visible on demand (i.e. handle visibility is controlled with a single key press) and act as intuitive signifiers that flexibly and accurately guide the ROI selection process. Both data independent (i.e. blended blob envelopes) and data dependent (i.e. GPU region growing) voxel selection is supported within the same 3D widget UI, allowing for fast and flexible volume exploration and ROI selection in 3D images ranging from relatively clean CT scans to noisy images such as MR scans.

2. A GPU region growing algorithm [7] that is controlled via the 3D widget UI and constrained by super-ellipsoid blobs, enabling the user to precisely select connected regions of voxels. This technique supports the highly-efficient real-time selection of individual objects, parts of an object, or objects with complex geometry and topology - such as artery networks.

3. An "open-view" capability that supports real-time exploration, examination and selection of occluded structures. An auxiliary adjustable cutaway "lens" can be attached to the camera at one end and the blob tool at the other. This capability allows the widget-based UI to overcome problems with viewing and selecting deeply "buried" or hidden structures. The adjustable cutaway lens design creates an effective perceptual depth cue of the cutaway region by orienting the cut surfaces towards the viewer.

4. A user study comparing the 3D widget-based "blob tool" UI, the previously developed "surface paintbrush" style UI and a standard cylinder "*screen-space paintbrush*" UI for defining a ROI. Surface paintbrush style UIs are well-known technique for intuitively selecting and removing outer region "layers" of voxels belonging to a specific tissue type, such as skin. However, due to the nature of the painting style they simulate, they may not be optimal for enveloping (i.e. selecting for highlighting or removing) structures with varying thickness. A 3D widget-based UI, on the other hand, is more volume-oriented but may require more user interaction to position and manipulate. Finally, the standard screen-space paintbrush is familiar, highly intuitive and often efficient for painting and thereby selecting 3D objects. However, for fast modification of the ROI to support volume exploration, the blob tool is more flexible than the surface and screen paintbrushes.

5. A second part of the user study compares a 3D widget-controlled, blob constrained UI (i.e. the blob region grow tool) for GPU region grow selection to a non-widget UI design that simulates the GPU region grow selection interface in a popular and freely available volume rendering package [8]. The goal of this part of the study is to determine if the interactive widget-controlled region growing algorithm offers any advantages in terms of performance, accuracy and controllability.

1.3 Thesis Outline

Chapter 2 provides a review of techniques for interactively visualizing and exploring 3D medical images. An explanation of TF-based volume rendering is provided followed by a brief review of existing TF based approaches. A more detailed review of 3D ROI selection algorithms and their associated user interaction techniques is then presented.

Chapter 3 presents implementation details of the system, including an overview of the user interface, the mathematical formulation of super-ellipsoid blending, pictorial descriptions of the various GPU-based program algorithms (known as *shaders*) and data structures used.

Chapter 4 presents quantitative and qualitative results in the form of a user study and a series of contextual view generation experiments in order to test the hypothesis that a combination of a 3D widget UI and blend-able super-ellipsoids is an effective visualization technique.

Chapter 5 summarises the thesis work and discusses the conclusions, future work, and improvements.

Chapter 2: Literature Survey

To gain insight into the information contained in an un-segmented volume image, it is necessary to be able to explore it from different viewpoints and to visualize anatomical structures “buried” inside it. One way this capability can be achieved is by using direct volume rendering and highlighting target structures using some sort of distinctive visual style. At the same time, it is often useful to render regions of the volume surrounding the target structures to help the user maintain visual context and aid their visual examination and quantitative analysis. As mentioned in the introduction, selecting features of interest through TF specification alone is often tedious and unintuitive and can still result in target features being occluded by other structures. This chapter will review context preserving feature visibility techniques that have been presented in the literature, many of which use TF specification along with an explicit user selection of a ROI. Before this review, the chapter will begin by providing some background on medical images and volume rendering via transfer functions. A summary and classification of the most commonly used types of TFs will be presented.

2.1 Volume Rendering: A Brief Review

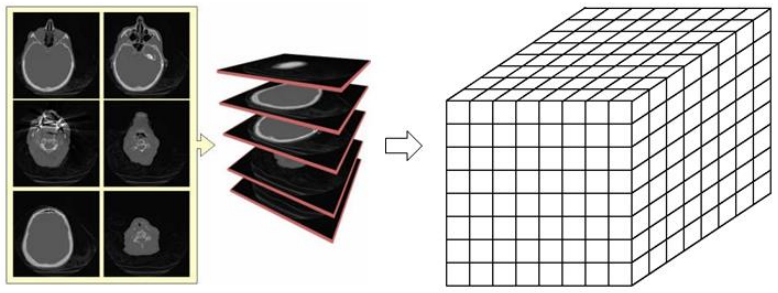


Figure 2: *A series of 2D MR image slices is stacked to form a 3D volume consisting of volume elements or "voxels".*

A 3D volume image is a contiguous set of 2D image "slices" acquired by *Magnetic Resonance Imaging* (MRI), *Computer Tomography* (CT), or *Positron Emission Tomography* (PET), etc. The set of contiguous slices form a regular volumetric grid (Figure 2). Similar to a pixel (i.e. a picture element) in a 2D image which can be visualized as a small square with an associated scalar value, each volume element, or *voxel*, can be visualized as a small cube with an associated scalar value (or *intensity*). These scalar values are the result of a scanning process and each value represents the measurement of signal intensity, such as x-ray absorption in the case of CT scans. For example, a CT scan creates a set of X-ray images taken from different angles. Computer processing is then used to create cross-sectional images, or slices, that form the volume.

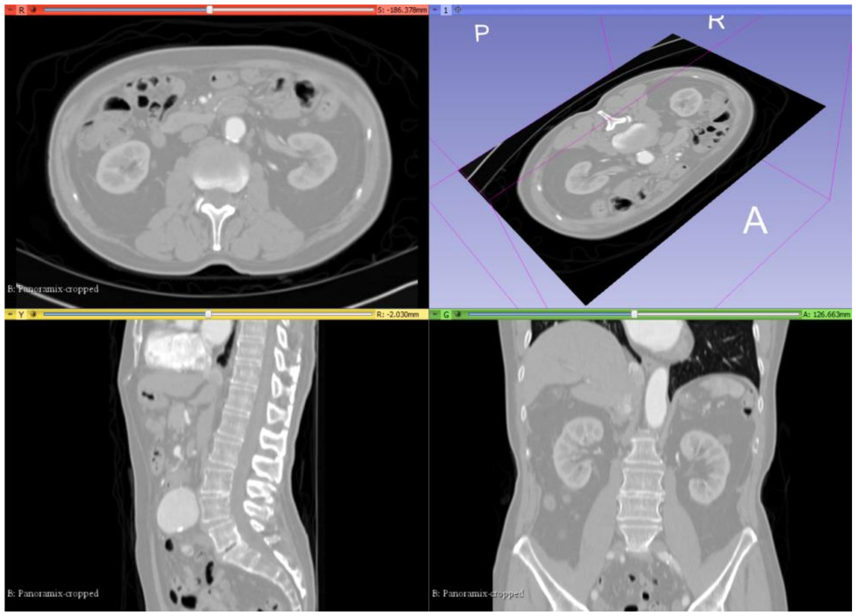


Figure 3: *Using multiple 2D projections of the volume to aid in the process of navigation and ROI selection in 3DSlicer* ***[9]****.*

As mentioned in the introduction, to visualize the data stored in the volume image, the scalar voxel intensity values must be mapped to a visual representation. The most common and widely used representation is a 2D image where the voxel values are mapped to a grayscale. Several standard 2D views (Figure 3) are typically provided corresponding to standard imaging planes: XY (axial), XZ (coronal) and YZ (sagittal). While these standard 2D slice views are still heavily used by radiologists and technicians, they require the mental reconstruction of a 3D anatomical structure from 2D projections. The user interface is also often made more complex as regions of interest must be marked on the 2D slice planes. Oblique slice views (Figure 3, upper right) are also often provided to allow radiologists to create approximately orthogonal cross-sectional views of curving anatomical structures and more easily perform this mental reconstruction and region of interest specification.

The multiple 2D views, however, often complicate user interaction and make it difficult for the user to understand the spatial relationship between the various 2D and 3D views. Despite the extensive training of medical image specialists to understand the 2D slice views, a 3D rendering of the volume image is cognitively simpler to understand and displays a large amount of perceptual information familiar to humans. Therefore, examining structures directly in this space is now standard practice. For this reason, user interfaces for interactively viewing and manipulating 3D data using traditional 2D mouse input device is a heavily researched field.

As mentioned in the introduction, rendering a 3D view of the anatomical structures buried in the 3D volume on a 2D screen window is now commonly performed using *volume rendering* and expert users manipulate TFs via GUI’s to control the visual output of the volume rendering algorithm. Direct volume rendering is a computationally intensive task that may be performed in several ways. In the next section, a volume rendering technique based on ray casting will be outlined. It should be noted that volume rendering is just one type of visualization technique that is used to create 2D projections of discreetly sampled 3D datasets. It is fundamentally different from another commonly used approach which is based on generating an intermediate surface representation - such as connected triangle meshes - from the 3D scalar field and then rendering these surfaces.

2.2 Ray Casting

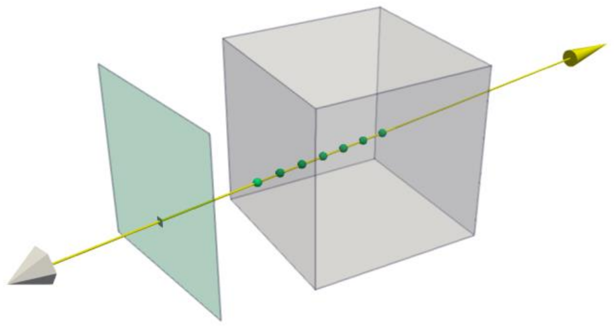


Figure 4: *Rays are cast through a volume and field values are sampled at regularly spaced intervals* ***[10]****.*

Ray casting is an image-based volume rendering technique. Rays are cast from the current view position through each screen pixel and traverse the volume (Figure 4). At (typically) regular intervals along the ray, the scalar intensity value of the volume is sampled. As the sample point may be between voxels, interpolation - for example tri-linear interpolation - is used to calculate an accurate field value. Higher-order interpolation can also be used for improved accuracy at the expense of additional computational cost. A user-defined TF is then used to map the intensity value to a *RGB* color and an opacity *A*. In addition, the intensity gradient is calculated at the sample point position using a finite difference approximation. This gradient represents the orientation of a local surface within the volume (i.e. a surface normal vector). A light source is typically positioned at the location of the viewpoint and this light source position, the sample point position, the normal vector and the color and opacity value outputted by the TF are used in an equation of a local illumination model [10] to *shade* the sample point position (i.e. determine a final color). The final color and opacity of the current sample point is then added to the current total color and total opacity for the ray and associated screen pixel. This *front-to-back* color and opacity accumulation process continues until the ray exits the volume or until the opacity reaches a pre-defined threshold (known as *early-ray termination*).

The color and opacity accumulation process can be described by the following recursive equation [10]:

$$\begin{cases} \boldsymbol{c}_i^* = \boldsymbol{c}_{i-1}^* + (1 + \alpha_{i-1}^*)\alpha_i \boldsymbol{c}_i \\ \alpha_i^* = \alpha_{i-1}^* + (1 + \alpha_{i-1}^*)\alpha_i \end{cases} \tag{2.1}$$

where $\boldsymbol{c}_i = (r_i, g_i, b_i)$ and α_i (alpha) are the current color and opacity, respectively, at the sample point $\boldsymbol{p}_i$, and $\boldsymbol{c}_i^*$ and $\alpha_i^*$are the accumulated color and opacity. More generally, Equation 2.1 can be viewed as an *alpha compositing* operation that combines the colors of the sampled ray points using the alpha opacity values as weights to achieve partial or full transparency. Equation 2.1 is a discrete approximation to the continuous light emission-absorption volume rendering integral equation which defines the light intensity after traversing the ray between two points on the ray. For an overview and introduction to volume rendering, the reader is referred to Engel et al. [11].

2.3 Transfer Functions

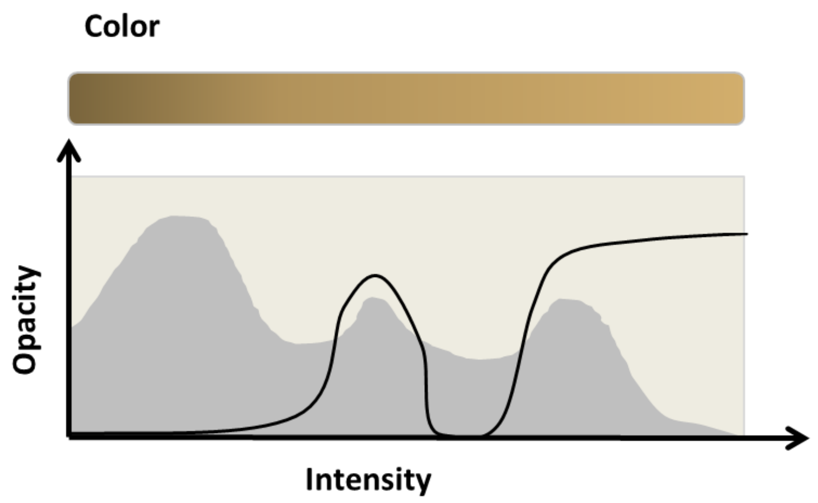


Figure 5: *(a) A 1D TF specifying color and opacity[8] values. The gray background represents the histogram of binned scalar voxel values.*

This section provides an overview of TFs in volume rendering. For a recent and complete review, the reader is referred to the recent survey paper [1]. The simplest form of a TF is a 1D function that maps scalar voxel values to color and opacity (Figure 5). Polynomial curve widgets are used to interactively assign voxel intensity ranges to a color and opacity. The polynomial curves can be piecewise linear or higher-order. Typically, a histogram is pre-computed in which the horizontal axis corresponds to every scalar intensity value found in the volume image and the height above the horizontal axis depicts the number of voxels with that intensity value (i.e. the frequency of occurrence). The gray background in Figure 5 is a visualization of the histogram. The user can manipulate (using a GUI) the curve and form a peak around a range of voxel values. The vertical axis of the TF represents the opacity assigned to a curve x-axis value (i.e. a voxel intensity), with

[8] In computer graphics opacity describes the level of a material's impenetrability to light.

the opacity varying from 0 to 1. A color band at the top of Figure 5 can be manipulated, via GUI controls, to assign an *RGB* color to a voxel intensity value. This 1D TF GUI interface is similar in most volume visualization software packages. For example, Figure 6 shows the TF interface for ImageVis3D [12].

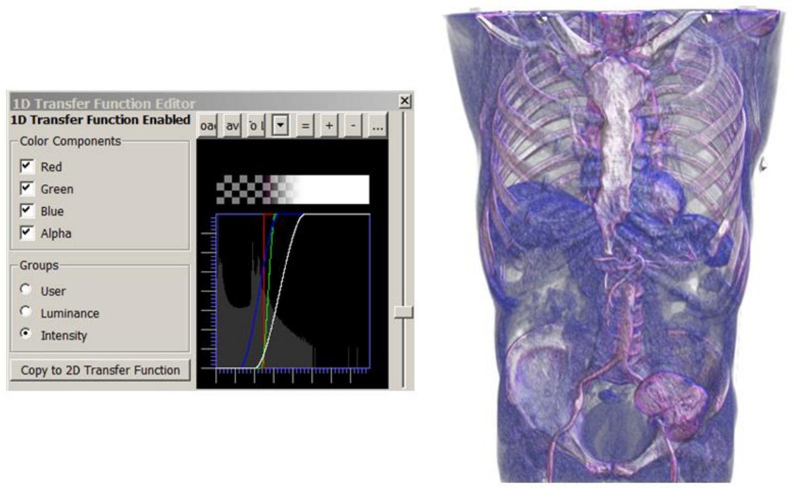

1D Transfer Function Editor
1D Transfer Function Enabled
Color Components
Red
Green
Blue
Alpha
Groups
User
Luminance
Intensity
Copy to 2D Transfer Function

Figure 6: *An example of a 1D TF and the resulting volume rendering of a CT volume [13] in ImageVis3D [12]. Color and opacity are assigned to voxel ranges using curve widgets (for example, the red curve in the figure). The gray background underneath the curves represents the histogram of binned scalar voxel values.*

Manipulating a simple 1D TF can be sufficient to generate a desired view of a data set, especially if the data is relatively noise free. However, it is often the case that such a TF cannot be used to visually separate some types of tissues (for example, arteries and bone) due to their overlapping scalar intensity values. Furthermore, medical data sets are derived from a scanning process and therefore the measurements are often noisy. Finally, partial volume effects[9] and intensity non-

[9] The partial volume effect is the lack of contrast in an image between two neighboring tissues due to the finite resolution of the medical scanner. The result is more than one tissue type measurement contributes to the field value of voxels near the tissue boundary.

uniformity (known as field bias[10] [14]) can occur. These factors make it difficult to correctly label and therefore visually separate different tissue types with a 2D histogram and a 1D TF. One possible reason for this difficulty is there is no local spatial correlation between the features in the histogram and specific anatomical structures in the volume.

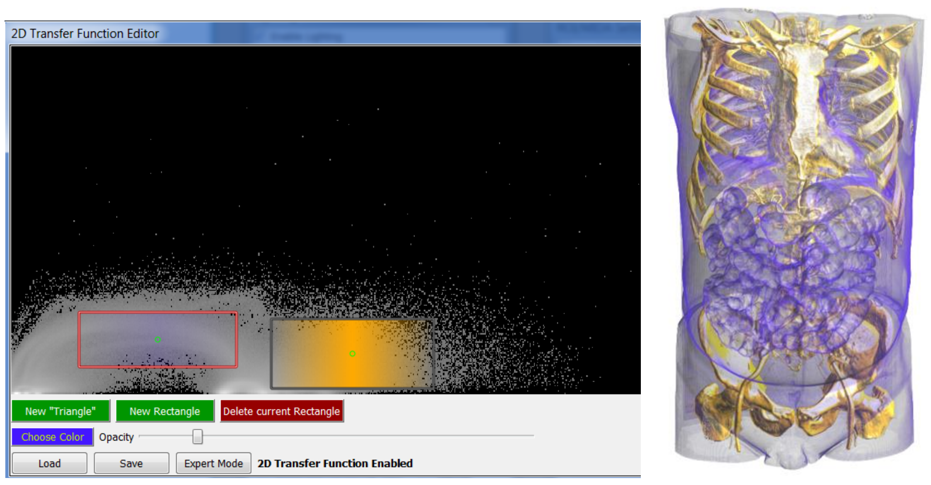


Figure 7: *A 2D histogram of voxel intensity values vs intensity gradient magnitudes in ImageVis3D [12]. Some regions and boundaries are selected and assigned different colors. The resulting generated image is on the right.*

Many 2D and multi-dimensional (MD) TFs have been proposed to overcome the problems with 1D TFs. For example, a common 2D TF takes as input not only the voxel intensity values but also the gradient magnitude at each voxel location [15]. Figure 7 shows a TF where the gradient magnitude is used to modulate the opacity such that interior homogeneous material regions are supressed, and material boundaries are enhanced, thereby improving visual perception of the volume rendering. A 2D joint histogram with voxel intensity values plotted along the horizontal

[10] Field intensity non-uniformity arises from the imperfections of the scanning process and results in a variation in the intensity of the same tissue at different location within the image.

axis and gradient magnitude values on the vertical axis is shown in the left of Figure 7. The end regions of an arch in the histogram visualization correspond to the homogeneous regions while the top parts of an arch represent maximum gradient magnitude regions and therefore correspond to material boundaries. The rectangles in Figure 7 represent widgets that allow the user to interactively select materials and material boundaries and assign to them color and opacity. Unfortunately, Figure 7 also illustrates how adding dimensions to a transfer function UI adds complexity and comprehension difficulty. Rectangular regions must now be adjusted to edit the visualization and the 2D joint histogram is not as intuitive as the histogram used in the 1D TF.

One of the problems with 2D TFs utilizing gradient magnitude is that material values and gradient magnitude values can still overlap and selecting features on a 2D histogram can be unintuitive. Therefore, much research has been done using MD TFs that consider additional voxel intensity (first and higher-order) derivative attributes, as well as other types of computed data attributes. However, the resulting MD TFs create difficult user interface design challenges both in terms of interaction and comprehension, as is suggested in Figure 7. Examples of data attributes are the use of curvature measures [16], voxel intensity statistics that characterize local neighborhood around each voxel [17], and the use of scale by computing a per voxel scale field characterizing the size of a local feature [18]. Other researchers attempt to address the TF user interface visual complexity issues for MD TFs by aggregating the extra attributes. For example, clustering algorithms can be applied to the histogram of voxel intensity values [19] and the TF interaction and visualization can be simplified such that the user selects and weights these clusters to generate the volume rendering. Other researchers simplify TF design by finding meaningful structure in the data, for example by using machine learning techniques on MD data attributes and mapping the lower dimensionality structured information to each voxel in the data set [20]. The TF is then applied to this map.

2.4 Focus plus Context Views - Utilizing Spatial Information

In summary, a large amount of research has been carried out to create UI's and TF's that enable medical professionals to generate insightful volume renderings of complex volume images, and much success has been achieved. In general, many 1D TF's and 2D TF's do not utilize global or local spatial information. However, how much spatial information is needed is perhaps task dependent [1]. For example, for tasks such as surgical planning for a brain tumor, it is beneficial to allow the surgeon to spatially localize and highlight target features or anatomical structures so that they can be viewed and measured with respect to surrounding structures. Formally, focus plus context visualizations attempt to visually combine a user-selected local spatial region of primary interest (the focus) with the surrounding information — or context — into a single display [6]. In a general sense, the focus region is differentiated from the context through the use of space, opacity, and color et cetera. In the context of volume rendering, a separate TF is used within the focus region, providing the possibility of a simpler UI design.

One of the most common focus plus context techniques is to make occluding objects semi-transparent thereby revealing hidden objects [21] [22]. For surface mesh data, context can be preserved by reducing the transparency of occluding objects according to the distance to the outline of the transparent object, where the outline typically consists of silhouette lines and is therefore view dependent [21]. For un-segmented volume data, the opacity of a sample point along a ray can be controlled by a function of shading intensity, gradient magnitude, distance to the viewer, and previously accumulated opacity [23]. Two user settable parameters allow the user to control the depth of the transparency effect (higher values reveal more of the volume interior) and the sharpness of the transition between transparent and visible regions. In general, the use of transparency is limited – it does not provide a strong visual cue of the depth of the hidden objects and it can be visually confusing if there are multiple overlapping layers of semi-transparent surfaces. The use of a distance function as part of the overall TF specification does take spatial information into account. However, the distance functions are typically radial in nature and do necessarily provide much local spatial control.

In [24] a voxel classification scheme is used based on the ambient occlusion of voxels. Ambient occlusion is equivalent to finding the centroid of the weighted histogram of intensities around that voxel. The authors argue that most volumes (e.g. CT or MRI) contain occlusion patterns that encode the spatial structure of features within the volume. A 2D TF is designed to incorporate the ambient occlusion information, leading to better control over the separation of interior features from exterior occluding features and from neighboring features with a similar intensity profile. Another advantage of the ambient occlusion information is it is easily understandable by medical professionals.

2.4.1 Focus plus Context Views using Explicitly Defined ROIs

Many researchers have developed techniques to generate focus plus context views where the focus region (i.e. the ROI) is more explicitly defined with a convex shaped lens region [25] [26] [22] [27] [5] [28]. The lens geometry is often a cylinder, sphere or cone or some other compactly defined mathematical function. For example, Zhou et al. [25] used focal region defined by a center, radius and a distance function where the opacity of a voxel is based on its distance to the focal center. The use of masking also enables the effect known as a “Magic Lens” or “Magic Lantern” [27] where the voxels inside the lens are ignored or highlighted. For example, Monclus et al. [27] uses a second TF to visualize a cone-shaped ROI in a way that is distinct from the surrounding material. Similarly, Ropinski et al. [29] uses a 3D convex shaped lens to define and render a volumetric focus region using non-photorealistic rendering. Tappenbeck et al. [30] uses distance-based transfer functions to hide or emphasize structures based on their distance to a reference structure. Kirmizibayrak [31] uses a polygonal object to define the boundaries for the focus Magic Lens sub-volume and also supports multi-modal rendering within this sub-volume. Burns [32] defines a cutaway region using two user-defined angles, both measured with respect to viewing direction ray. The larger of the two angles separates a base opaque region (the context) from a focus region. The smaller angle controls the transition of opacity within the focus region. Any voxels along the rays cast from the eye, where the angle between the ray and the viewing direction ray is less than the smaller user-defined angle, are rendered transparent (clear). The opacity of other voxels in the region between the clear region and the base region is smoothly transitioned to fully opaque. Similar to some of the work presented in this thesis, Bruckner and Gröller [33] uses a 3D volumetric painting approach with a 3D Gaussian lens.

Lenses realized with super-ellipsoids have also been used by other researchers, albeit often in more restricted ways, for volume image exploration and ROI selection. Similar to the super-ellipsoid lens presented in this thesis, Luo et al. [28] uses a super-ellipsoid distance function to define their focal probe region. Within this region, a different rendering style may be used than in the surrounding context. They also incorporate a view-dependent cone region to cutaway occluding voxels in front of the focal probe. Radeva et al. [34] also uses a super-ellipsoid lens and incorporates an on-demand 3D image slice view into the lens. The image slice supports the rendering of a different modality image, providing the ability to simultaneously display mixed modalities. Both of these research works share similarities with the super-ellipsoid lens presented in this thesis. However, in this thesis the blob tool can act as a “lens” and multiple super-ellipsoid paint blobs can be deposited and smoothly blended to define a complex–shaped region (Section 3.2.1). As mentioned in the introduction, this work can be considered an extension of [5]. However, unlike the work presented in [5], in this thesis the super-ellipsoid blob tool can be used to constrain and guide a region growing operation to segment complex visible structures such as arterial trees and uses a more advanced widget-based UI.

Another issue with a convex lens approach is the target region/object may be occluded (i.e. hidden) by other objects. In this thesis, a user-controllable view-dependent auxiliary lens is attached to the blob tool and generates cutaway views in front of the blob to provide enhanced depth perception of target objects. Luo et al. [28] also employs this strategy although in this thesis the shape of the auxiliary lens is not restricted to a cone. The lens shape is configurable and is designed to provide depth cues via the cut surfaces of the occluding objects such that the relative position and depth of the target object is easier to perceive (see Chapter 3, Section 3.1.5).

2.4.2 Selecting a Complex-Shaped ROI as a Focus Region

Interactive selection of complex-shaped 3D ROI selection techniques can be categorized in several ways. One categorization is data dependent techniques versus data independent techniques. An interactive data independent technique specifies a ROI spatially (i.e. geometrically). A data dependent technique uses data attributes, such as voxel intensity or voxel gradient magnitude, to label voxels as part of the same structure. This segmentation process is a heavily researched field in the world of medical image analysis [35] [36] [37] and advanced segmentation algorithms are often required to select structures in noisy images. Advanced segmentation algorithms are beyond the scope of this thesis. Instead, simpler semi-automatic highly-parallel GPU-based segmentation algorithms, such as voxel region growing, can be used with great effect in volume rendered images such as CT scans and MR scans to quickly define a ROI and generate a contextual view for previewing and examining of a target region. These techniques may be sufficiently accurate to perform a visual analysis of structures by quickly and iteratively generating contextual views from different viewpoints and of different spatial cutaway shapes and extents. Measurement tools may then be used to establish, for example, the 3D position and extent of an aneurysm. As mentioned above, for more detailed measurements of specific target structures in noisier volume images such as MR scans, advanced segmentation algorithms – both semi-automatic and automatic - are typically required. However, as mentioned in the introduction, a user-defined “envelope” region that surrounds a target structure can often be used to improve the robustness and efficiency of semi-automatic algorithms [4].

Geometric techniques that are primarily based on user interactive spatial specification of a ROI can take many forms. The advantage of these techniques is that they can be applied to any volume, regardless of noise. For example, a 3D ROI can be created by defining a 2D ROI on successive image slice planes and then merging them to form the 3D ROI [4].

Geometric approaches to focus region specification, while noise independent, places the emphasis on the user to navigate to a region of interest and define an envelope. Hence, the effectiveness of the spatial localization hinges upon the user interface. A simple, intuitive UI that is flexible enough to define desired regions is the goal. A common strategy to deal with the occlusion problem is to cut away, deform, or make the occluding parts of the volume semi-transparent in such a way that the features of interest become visible.

2.4.3 Categorizing Complex-Shaped ROI Selection Techniques

Techniques for interactive geometric specification of a ROI can also be categorized based on the underlying interaction metaphor used. Such a categorization can be useful for gaining an overall insight into the task. *Outlining* (or sketching), *painting*, *sculpting*, and *deforming* are among the most common metaphors used. In the following paragraphs we briefly describe these interaction metaphors and provide illustrative examples from the literature. It should be noted that many of the research works referenced below contain aspects of two or more of the categories within their interaction metaphor. Nevertheless, for simplicity, each referenced work is slotted within a particular category.

2.4.4 Outlining/Sketching

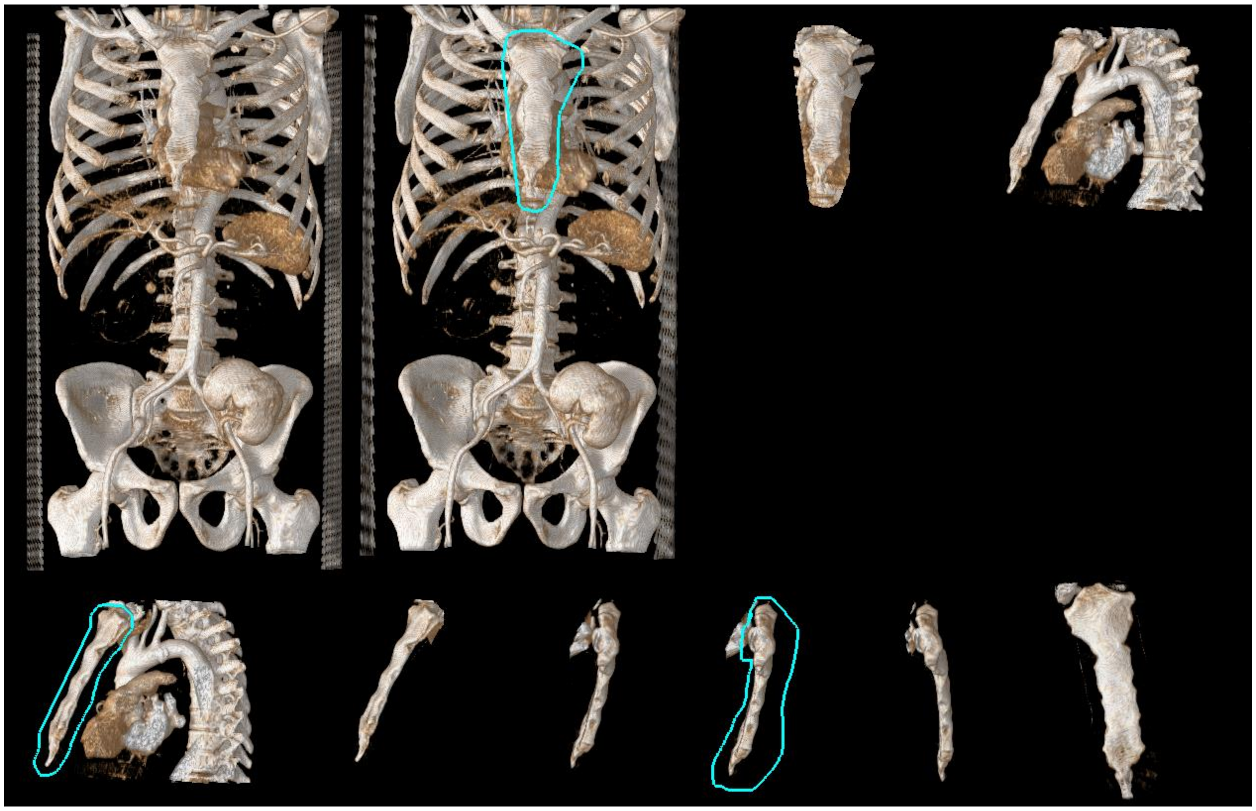


Figure 8: *Outlining interaction metaphor used in LiveVolume [8] to outline the sternum. From left to right: 1. Original 3D visualization. 2. Create an outline of the sternum using LiveVolume's [8] region select tool. 3. Voxels outside of the contour are removed. 4. The scene is rotated to reveal the additional voxels selected by the contour. 5. Refine selection by creating another contour around sternum. 6. Repeat process until only sternum is selected.*

In a tracing or outlining interaction metaphor for ROI selection, the user draws/outlines an accurate contour on the screen, or on a plane defined within the volume image, or even directly on the object surface, to delineate a 2D region [38] [39] [8] [40] [41]. To create a 3D region, several techniques can be used. The contour can be extruded – i.e. copied and translated in depth and then connected to the previous contour. The extrusion direction is often in a direction orthogonal to the viewing plane. Alternatively, when outlining on the screen, the screen pixels contained inside the contour region can be used as a "mask" in screen space and any visible voxels encountered along rays of the volume ray caster during volume rendering are projected onto this screen space. Any voxels inside the mask region are selected. Finally, tracing/outlining may also be performed on multiple 2D slices [42] [3], either manually or semi-automatically, where a subsequent contour stitching is required to connect adjacent contours and form the 3D ROI. Similarly, in a contour *sketching* interface, the user quickly draws line/curve strokes on the screen [40], on the data surface [43], or on a series of 2D slices [4]. These strokes may overlap [44]. An algorithm then automatically connects these primitives to form a contour. Similar to outlining, these contours are then connected [4], or "inflated" [44], to form a 3D surface envelope. Sketching actions tend to be quick, rough approximations and typically allows a user to introduce protrusions and bumps and other spatial features on the 3D ROI in a progressive manner.

While tracing/outlining/sketching a 2D region on the screen/plane is simple, intuitive and precise, extending this region to 3D using a 2D input device (e.g. a mouse) so that it selects target objects in a 3D volume rendering is a more complex task. The user begins by rotating the 3D volume rendered view of the data such that the target region/object to select is visually separated from the (Figure 8) context region. The user traces around the object and extracts the region inside the contour. This scene rotation and outlining process is repeated, progressively refining the 3D ROI. While an individual tracing action itself is fast and precise, the increased cognitive load[11] on the user of rotating the volume to find views that separate the target ROI from the rest of the volume, along with the subsequent tracing action, can result in less efficient and more difficult selection in

[11] Cognitive load refers to the mental effort imposed on *working memory* in any one instant.

some scenarios. Controlling the depth of the 3D ROI is also problematic. Most screen space techniques, for example, simply set the depth of the 3D ROI to the far side of the volume image and the user can only see the selected 3D ROI shape after by scene rotation. Furthermore, the efficiency of the tracing interaction itself depends on the complexity of the target shape. If a mistake is made, the current outline is typically discarded and a new one generated. That is, there is no facility inherent to outlining for 3D ROI shape editing. Outlining on multiple slices may also complicate the interaction - especially if the slices are arranged on a curving extrusion path or if some of the slices are orthogonal to others [42] [3]. Finally, outlining/sketching plus stitching/extruding 3D shapes with complex topology is also inherently difficult and therefore often not supported. The above issues may negatively affect ease of use, control and flexibility of the technique. Fundamentally outlining/sketching is progressive, multi-stage process where the selection is observed at the end of each stage to determine if it is complete. Thus, while this approach is useful for selection in many scenarios, it is not as appropriate for fast 3D ROI refinement required for volume exploration. Efficient volume exploration via selection requires a technique that supports fast, fluid instantly observable changes to the 3D ROI.

2.4.5 Volume Clipping and Sculpting

A sculpting interaction metaphor simulates cutting/sculpting tools with convex-shaped tool “tips” [45] [46], such as spheres, cylinders, rectangular blocks, and super-ellipsoids. Any voxels inside the tool tip are selected and sculpted away. Progressive cuts can be achieved, analogous to a sculptor with a chisel carving away pieces of stone. That is, rather than selecting voxels, sculpting typically attempts to reveal structures inside a volume by physically removing parts of the volume that otherwise would obstruct them. Typically, sculpting style interactions are performed directly in the volume space, sometimes using a high DOF input device to position and orient the tool tip[12] [45] or using a 3D widget interface if a mouse is used. A classic example of sculpting is the work of Weiskopf et al. [45] to cut away parts of a volume by using various geometric primitives and depth test algorithms. Sculpting style interactions are familiar and easily understood by users and this interaction technique can be quite effective when cutting away regular shaped regions. One problem with sculpting interactions is hidden target voxels can be inadvertently removed resulting in an undo-redo sculpt sequence. Consequently, this scenario may result in the user using small tool tips when accurate ROI boundaries are desired in order to avoid undoing and re-sculpting. Another issue is perception – the progressive cutaway can make it difficult to perceive the extent of material removed – especially if the material has similar appearance to the surrounding structures.

2.4.6 Deforming

Directly deforming (i.e. pushing and pulling like kneading dough) occluding parts of the data volume out of the way, or deforming a supporting geometric model that such that it envelopes or separates occluding regions, is another type of ROI selection interaction metaphor [47] [48] [49]. Typical deforming actions include pushing, pulling, stretching, or peeling [50]. The underlying data volume or supporting geometric model is often treated or modeled as a flexible material. Generating a ROI using a deformation approach can be quite effective in some ROI selection scenarios. The pushing/pulling actions of the model are easy to perform, intuitive and can efficiently remove large occluding volumetric regions, especially when the flexible geometric

[12] High DOF input devices are not explored in this thesis.

model is constructed as a deformable clipping surface [47]. The deformable material can be interactively "edited" to quickly change the ROI. However, deforming a supporting geometric model to select individual objects in the volume - especially curving, twisting, and/or elongated objects, or objects with varying thickness - may negatively affect ease of use and control as well as efficiency. In general, deformation may be better suited for separating/enveloping *regions* rather than specific objects.

2.4.7 Painting

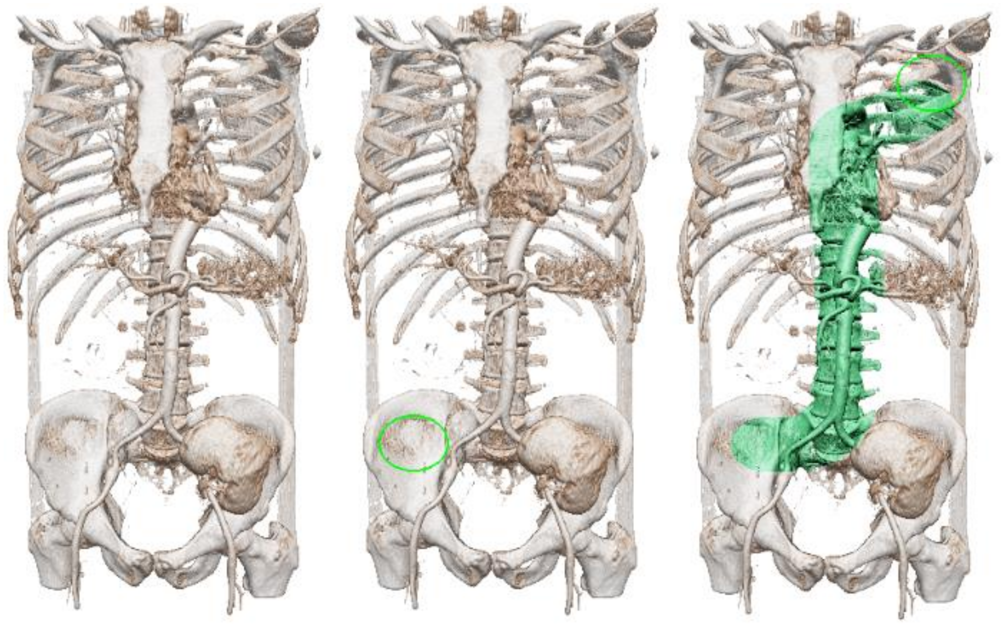


Figure 9: *An illustration of the screen painting technique used in the system. From left to right: 1. Initial scene 2. Screen painting mode enabled. 3. Perform paint stroke.*

Selection via painting typically defines a "brush" as a circular area along with a user-specified depth/thickness of the brush to form a cylindrical region. Painting actions are typically constrained such that the brush slides along some surface - most often an iso-surface of the data generated by the TF specification - mimicking real world painting [31] [7] [33] [51]. Painting is also commonly performed directly on other surfaces such as the screen plane (Figure 9), an image slice plane or some other user-defined painting plane [5]. As the brush slides along a surface, paint is continuously deposited and voxels inside the painted region can be rendered semi- or completely transparent. Depositing paint can mean planting individual paint “blobs” of a user-settable thickness which automatically smoothly merge with overlapping blobs to form a thick layer of paint [33] [5]. Alternatively, as the circular brush slides on the iso-surface or on the screen, a continuous mask region is created. During volume ray casting, if any voxels encountered along a ray project onto the surface/plane such that they are inside the mask region, they are selected and highlighted using the paint color [7]. Painting style interaction is simple, fast and intuitive as the underlying painting metaphor is familiar and easily understandable. Selection techniques based on painting are also typically quite flexible and easily support complex shaped regions. Painting often works very well for removing *layers* of voxels or “thin shell” ROIs [7] such as the skull region in a CT scan. Painting readily supports a previewing capability by using the brush as a “lens” [33] [5]. However, painting on a data iso-surface requires relatively noise-free volume images (i.e. to generate “clean” iso-surfaces) and therefore is primarily useful only for CT scans. Otherwise, the “paintbrush” can inadvertently stick to small disconnected pieces of the iso-surface which can be quite distant from the target object surface. A second issue is 3D ROI depth specification. This issue is also true when painting on the screen or on a painting plane. If the target structure or region is of varying thickness, then predefined paintbrush depth specification cannot be used and frequent dynamic adjustments of paintbrush depth/thickness affect ROI painting efficiency. If ROI boundary accuracy is required, the user may need to erase/undo already painted regions that have “spilled over” into neighboring structures behind the ROI. One solution is to use smaller thinner brushes and paint over target regions multiple times, progressively removing “material” and progressively revealing deeper layers of the ROI [7]. This works well when the goal is to remove the occluding layers. However, for selecting deep occluded structures, the paint selection technique is not as applicable. For this reason, painting style selection is often combined with an interactive region grower or other segmentation algorithm that can select these deeper, occluded structures once parts of them are revealed via the painting actions (see Section 3.1.1).

2.4.7.1 Volume Painting Using Blended Spherical and Super-Ellipsoid Blobs

As mentioned in the introduction, in previous work Faynshteyn and McInerney [5] also used blended blobs to isolate, via painting, regions of interest. These blobs were primarily spherical in shape and the blending algorithm also generated a bulging effect when blobs were blended – sometimes resulting in unintuitive ROI selection. While super-ellipsoid blobs were also supported, the mathematical formulation of the super-ellipsoid and blending was not well-developed. There was also no open-view capability – making it difficult to select an occluded ROI. The result was that the technique was better suited to removing layers of tissue to reveal underlying structures. Viola et al. [33] describe a framework called *VolumeShop* that supports illustrative volume rendering techniques. It includes a *surface-constrained 3D painting* capability that uses a 3D Gaussian paintbrush to deposit Gaussian blobs which can be blended to form the 3D ROI. However, Gaussians have less shape control than super-ellipsoids and also suffer from unintuitive bulging effects where the blobs blend.

In this thesis, the combination of the 3D widget-based UI and super-ellipsoid blobs contains aspects of both sculpting and painting interaction styles. Similar to sculpting-style tools, the super-ellipsoid lens used in this work acts as a cutting tool or as a highlighting tool. Unlike most sculpting-style tools however, the super-ellipsoid blobs can be deposited and smoothly blended like paint, if desired, to create a smooth envelope of paint defining an arbitrarily shaped ROI. This feature allows the paint envelope to be used as an initialization of a segmentation algorithm, if desired. It is also useful to tightly envelope a target object to better control the volume rendering within. A widget is an indirect interaction technique (slightly less direct) but has certain advantages. In Chapter 4, we compare this approach to a iso-surface painting style selection as well as to a standard screen space painting approach. The advantages of 3D widget- based techniques are they can provide precision and control and if the visual signifiers (i.e. the handles) are well designed, then they are simple and intuitive to use. The main disadvantage of 3D widgets is visual clutter and the serial nature of the widget positioning.

2.5 Interactive Selection Techniques for Data Dependent Focus Regions or Image Features

Interactive segmentation algorithms such as GPU based 3D interactive region growing are a fast and effective way of selecting objects in relatively noise-free CT scans and MR scans. The algorithm starts from a user selected “seed” voxel as well as user-defined intensity range, and optionally a user-defined voxel gradient magnitude range. Voxels that are spatially adjacent to the seed voxel and are within the user-defined value ranges are selected. These “child” voxels then become new seed voxels and the algorithm repeats until no new child voxels are added. The algorithm can typically also be under interactive control (using a mouse movement, for example [8]) by enabling some control over the growing process. Furthermore, it is also common to support interactive shrinking by unselecting voxels back to a previous voxel set. Some implementations further allow for rough control over the spatially localized growing/shrinking of the selected voxels

[7]. Region growing is a noise sensitive algorithm so for noisier volume images, such as MR images, it is often not possible to generate an accurate segmentation, especially for structures buried deep within the volume. In these cases, more sophisticated semi-automatic or automatic segmentation algorithms are required.

The volume visualization package LiveVolume [8] also supports fast GPU region growing to select complex connected voxel regions. The UI is simple and effective – an initial point is selected, and the user uses the mouse to control the growing/shrinking of this region in 3D by moving the cursor away from, or towards this initial "central" point. This type of region growing control is compared to the widget plus super-ellipsoid constrained region growing of this thesis in the user study (see Chapter 4). Guo and Yuan [52] also use "region growing" techniques – i.e. based on topology (connectedness) of voxels values. In this work, all connected regions are preprocessed and organized into "branches" that can be instantly selected by simple painting on the voxels. Region growing algorithms are susceptible to noise and the preprocessed regions may inadvertently contain voxels from neighboring features or structures. Therefore, some user control is given up for automation and speed. Chen [53] [7] also supports fast GPU region growing. They use a sketch-based UI where the user sketches curves on the screen. These curves are projected into the data volume and any iso-surface voxels with a value that is within a user-defined range are selected as seed voxels. Reversing the growth is supported (i.e. region shrinking) and localized region shrinking is accomplished by sketching additional curves on the screen to indicate regions that should be shrunk.

The goal of Chen's work is primarily segmentation of a target object whereas the goal of the work in this thesis is primarily dynamic contextual view generation. The idea is to quickly select target regions and apply a separate TF to highlight or remove them. The widget UI is designed to interactively modify the selection to quickly generate contextual views from several viewpoints in succession. In this thesis, having the region growing functionality controlled by the same 3D widget UI as the geometric 3D ROI selection allows a user to precisely select regions of connected voxels that might be otherwise difficult and/or time-consuming to isolate using only the geometric approach. The two techniques are complementary.

2.6 Interactive Navigation and Manipulation Interfaces for 3D Images

Generating contextual views by utilizing spatially localized information within a volume rendering requires the user to define a local region. This commonly means that the user needs to quickly and easily interactively specify the position, spatial extent, shape, and optionally the orientation of the local region. Specifying these parameters requires the ability to move in and around the volume, known as navigation[13]. One of the primary navigation techniques for volume images is 3D rotation of the volume. This interaction is most commonly based on a virtual trackball such as the well-known *Arcball* [54]. Other than volume rotation, moving around in the volume image, by specifying the position of a focus lens, is also a primary interaction. Orienting and scaling the lens is also a common requirement.

There has been some research done in studying high degree of freedom input devices[14] for navigating, exploring, and selection in 3D medical images in a desktop environment [45] [51]. However, the mouse, a two degree of freedom (2-DOF) input device remains as the preferred

[13] For an overview of navigation in 3D virtual environments, the reader is referred to [62].
[14] A review of high DOF input devices is beyond the scope of this thesis.

choice due to its speed, high precision, ease of use, and accessibility. Comfort is also an important factor – when using a mouse, the user's arm rests on the desk and the large muscle groups of the shoulder, which are more easily fatigued, are not activated. However, since the volume image is a 3D space, a mapping must occur between the 2D input and the 3D output. Additional input in the form of modifier keys and/or the mouse wheel are often used to implement this mapping.

As described in the section on 3D ROI selection via painting, another technique for navigating in a volume image is to use the mouse to pick points on a volume rendered data iso-surface [5]. In addition, the surface normal vector can be used to orient a ROI delineator such as a convex lens. However, as mentioned, this technique is noise sensitive. While an iso-surface position specification can be used for translation and orientation, scaling of the ROI requires a separate input mode. In addition, only visible surfaces can be utilized, and it is difficult to pick points on the surfaces of thin structures such as arteries. Another well-known volume positioning technique is to use the user-controlled camera view direction along with a depth specification (via the mouse wheel, for example) to establish an oriented plane. The mouse can be used to pick points on this plane [5]. Faynshteyn and McInerney [5] use a combination of data iso-surface lens positioning and camera view plane lens positioning.

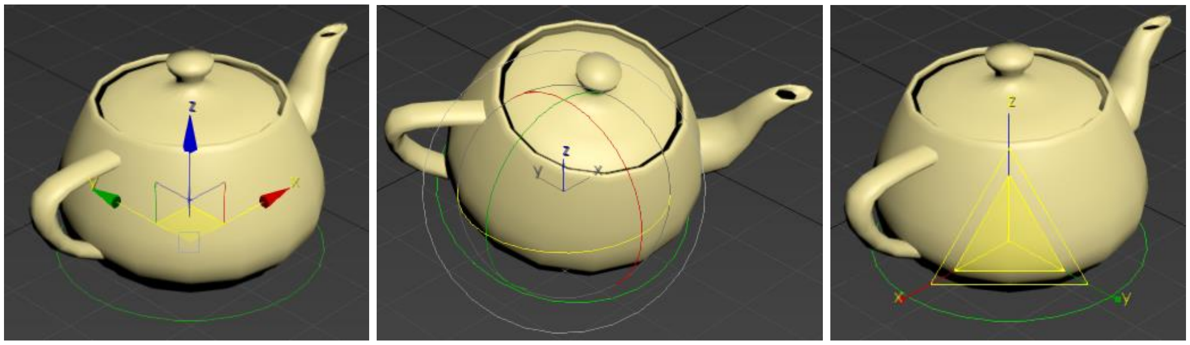


Figure 10: *Translational, rotational and scale widgets of the 3DS Max[15] modeling package.*

Another common technique for 3D volume navigation and manipulation is the use of 3D widgets. Widgets present to the user a proxy geometry in the form of "handles" (i.e. signifiers [55]) whose visual appearance suggest to the user their affordance (Figure 10). Translation, orientation and scale handles are presented as separate widgets or combined into a single widget. By selecting the handle and associating a shape primitive (e.g. a lens) with the handle, the primitive can be translated, oriented and scaled. The use of widgets is standard in modeling packages, game engine software, and visualization packages. Widgets are a precise and data independent 3D navigation and manipulation technique. The main problem with 3D widgets is visual clutter. In addition, the same strength of 3D widgets – data independence – can also affect the efficiency of the technique for translating and orienting a primitive (e.g. a lens) with respect to a target object. Control over scaling a lens is, on the other hand, quite efficient and intuitive. In this thesis a 3D widget UI for lens manipulation is proposed that also is integrated with data iso-surface positioning approach.

[15] 3ds Max, https://www.autodesk.ca/en/products/3ds-max/overview

The widget UI can be used alone or in concert with data iso-surface picking, enabling its use on even noisy data sets. The UI is detailed in Chapter 3.

Chapter 3: Methodology and Implementation

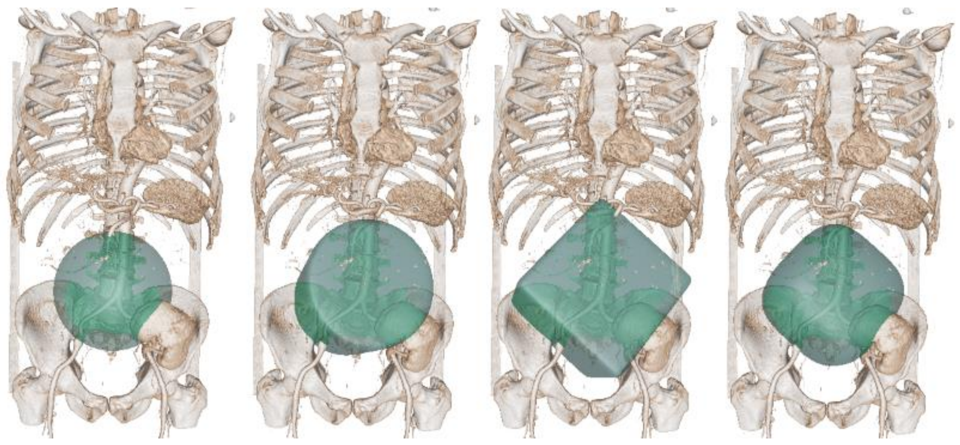
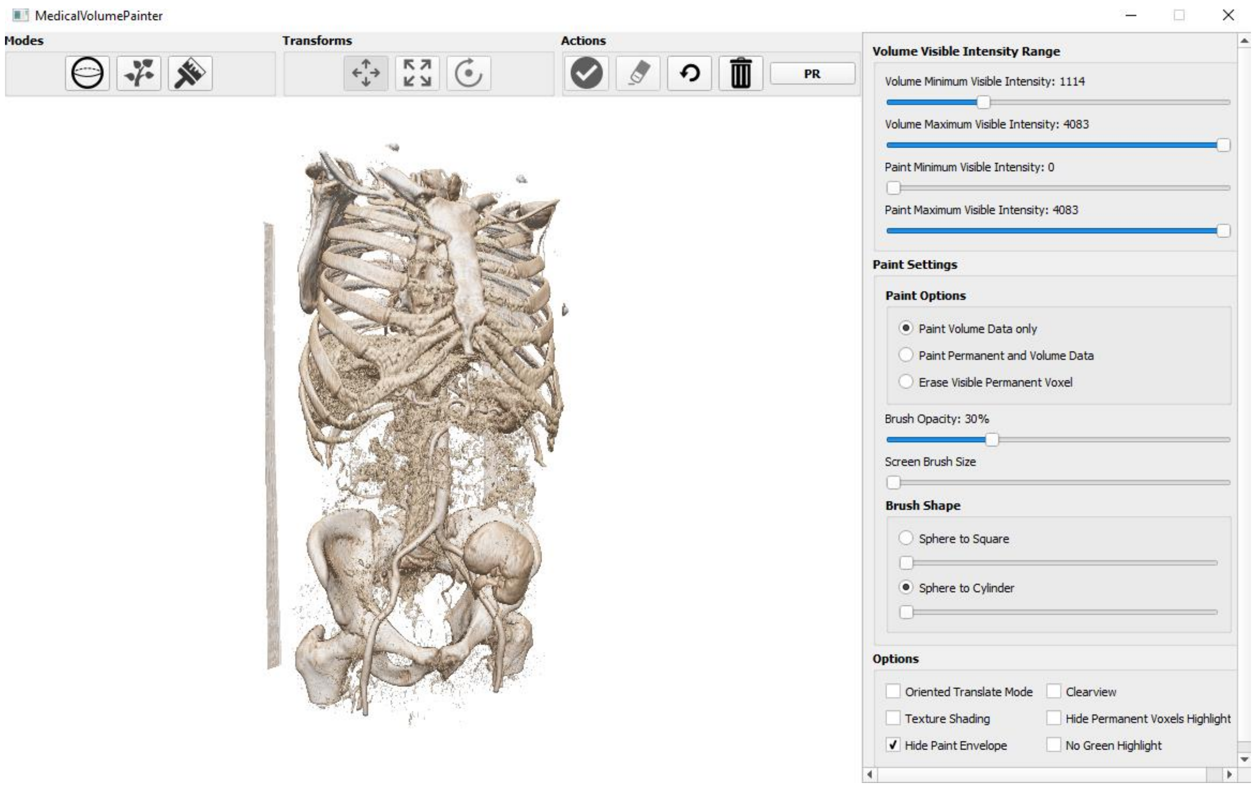


Figure 11: *An example of the range of shapes created by the super-ellipsoid blob tool. The Super-ellipsoid shape is controlled by a few intuitive parameters via GUI sliders. For example, the blob can take the shape of a sphere, cylinder, cube, or anything in between.*

In the previous chapter a summary of TF based techniques as well as a summary of 3D ROI selection techniques was presented. This chapter describes the mathematics and implementation details of the widget-based blob tool for volume exploration and ROI selection. The system combines volume rendering, a super-ellipsoid “blob tool” (Figure 11), a super-ellipsoid blending algorithm, and a super-ellipsoid constrained region growing algorithm with an intuitive and precisely controllable 3D widget UI to support the flexible real-time generation of focus plus context views. This chapter is divided into two sections; a description of the user interface followed by the implementation details describing the mathematics, algorithms and data structures used.

3.1 User Interface



Volume Visible Intensity Range

Volume Minimum Visible Intensity: 1114

Volume Maximum Visible Intensity: 4083

Paint Minimum Visible Intensity: 0

Paint Maximum Visible Intensity: 4083

Paint Settings

Paint Options

- [✓ Paint Volume Data only]
- Paint Permanent and Volume Data
- Erase Visible Permanent Voxel

Brush Opacity: 30%

Screen Brush Size

Brush Shape

- Sphere to Square
- [✓ Sphere to Cylinder]

Options

- Oriented Translate Mode
- Clearview
- Texture Shading
- Hide Permanent Voxels Highlight
- [✓ Hide Paint Envelope]
- No Green Highlight

Figure 12: *A snapshot of the graphical user interface of the volume rendering system presented in this thesis.*

The graphical user interface (GUI) (Figure 12) of the system was developed using Qt[16], a cross-platform application framework and widget toolkit for creating classic and embedded GUIs. The design of the interface is also loosely based off well-known 3D computer graphics modeling software such as Blender[17], and game engines Unity[18] and Unreal[19].

[16] Qt, http://qt-project.org
[17] Blender, https://www.blender.org/
[18] Unity, https://unity3d.com
[19] Unreal, https://www.unrealengine.com

3.1.1 Painting Modes

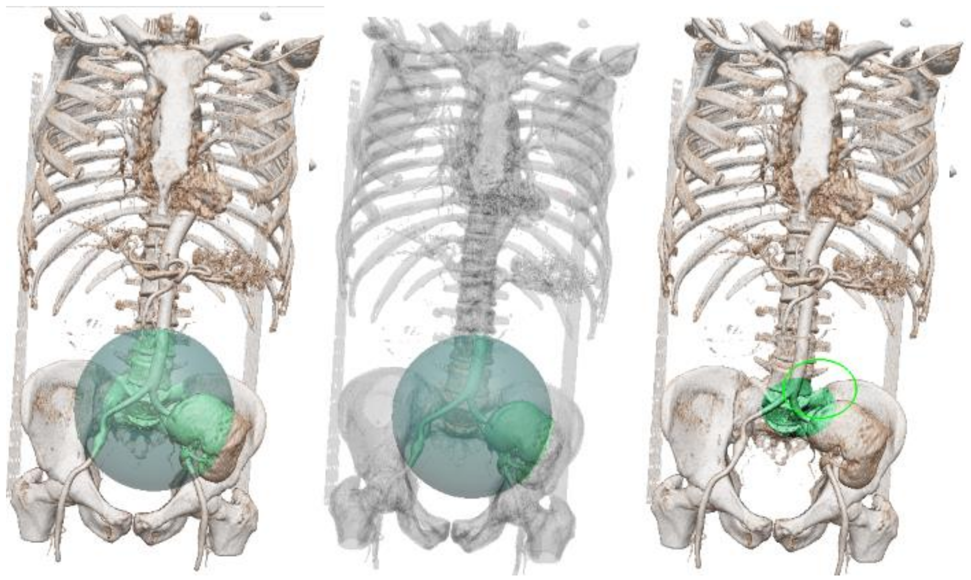
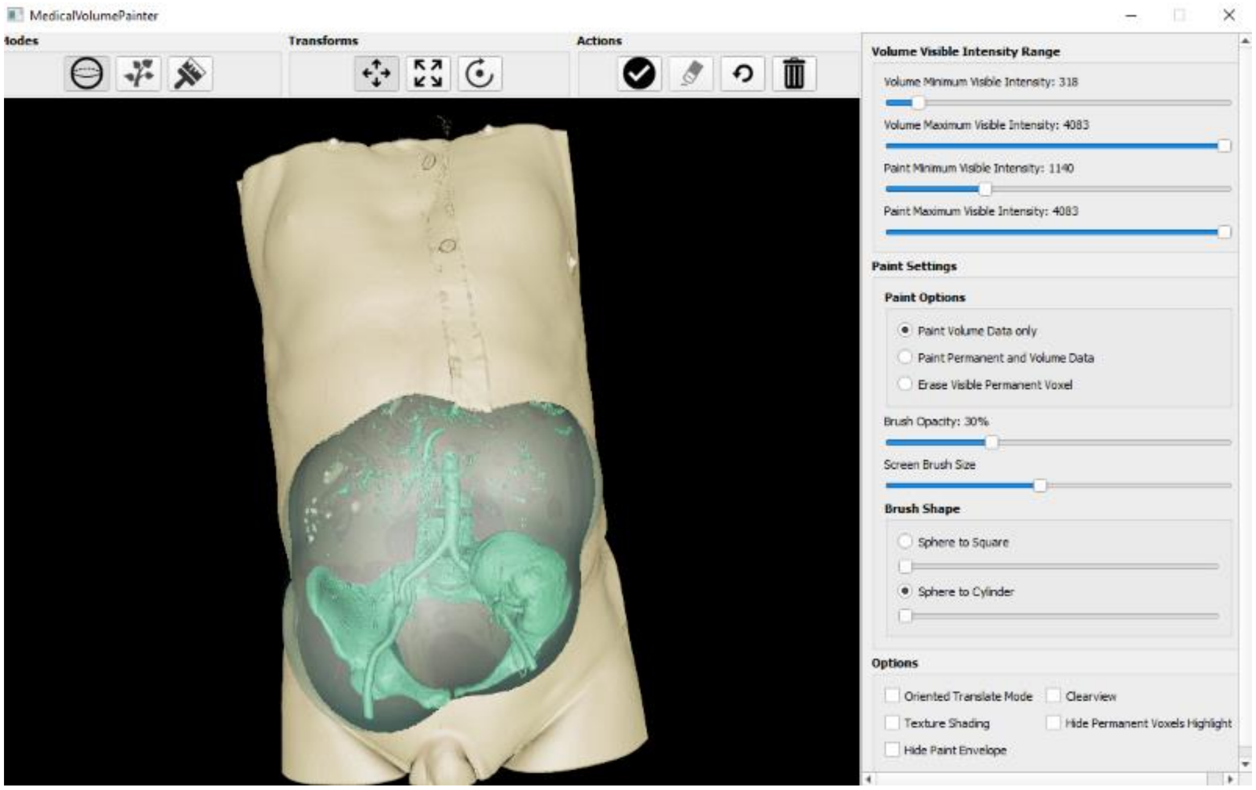


Figure 13: *The three painting techniques supported in the system. From left to right: blob tool, blob region grow tool, and screen brush. Note the different regions highlighted by each tool.*

The system primarily uses the super-ellipsoid blob tool for painting and exploration. However, the blob and/or paint can also be used to constrain a semi-automatic segmentation technique. In this system, a super-ellipsoid is used to constrain a region grow algorithm which provides users with additional control over how a ROI is selected. This tool is referred to as the blob region grow tool. A screen space painting technique was also added to the system for comparison purposes and is an additional quick and intuitive way of selecting, editing, and deleting areas of interest. When these techniques are combined, a user can interactively create contextual views for any dataset. Figure 13 illustrates the three different painting techniques supported in the system.

3.1.2 Voxel Intensity Visibility Range



Volume Visible Intensity Range

Volume Minimum Visible Intensity: 318

Volume Maximum Visible Intensity: 4083

Paint Minimum Visible Intensity: 1140

Paint Maximum Visible Intensity: 4083

Paint Settings

Paint Options

- [✓ Paint Volume Data only]
- Paint Permanent and Volume Data
- Erase Visible Permanent Voxel

Brush Opacity: 30%

Screen Brush Size

Brush Shape

- Sphere to Square
- [✓ Sphere to Cylinder]

Options

- Oriented Translate Mode
- Clearview
- Texture Shading
- Hide Permanent Voxels Highlight
- Hide Paint Envelope

Figure 14: *A scene with a ROI defined by the blob tool. The minimum and maximum visible voxel intensities for the blob region and the context region (skin, in this example) have separate controls and can be blended together seamlessly to create a focus plus context view.*

Users can control which voxels are rendered by the volume ray caster by manipulating GUI sliders to control the minimum and maximum voxel intensity visibility thresholds for the volume image and super-ellipsoid defined regions. In effect, the sliders represent a simple rectangular step function that defines the intensity range in which voxels are considered visible by the volume ray caster. Figure 14 illustrates how the sliders can be manipulated to alter the scene.

3.1.3 Blob Tool Manipulation

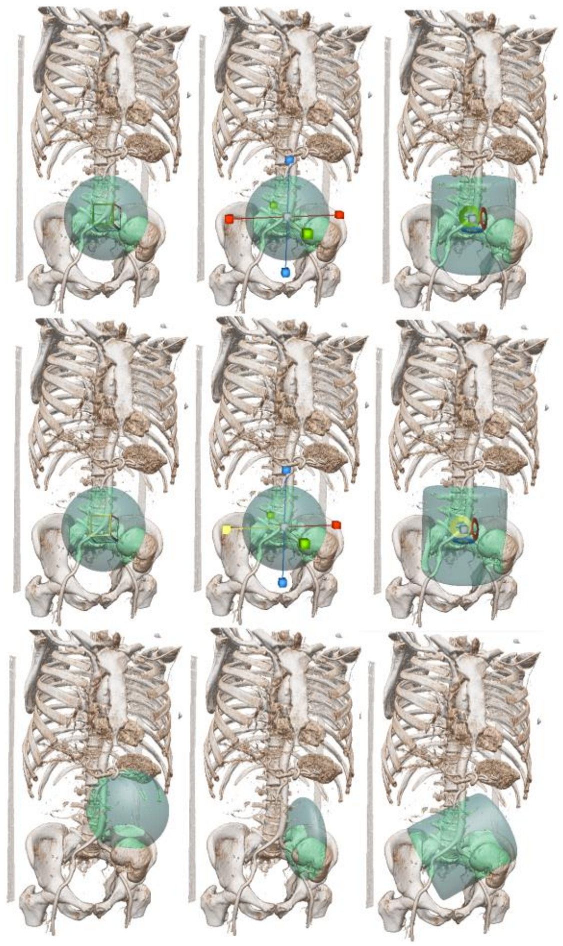


Figure 15: *Super-ellipsoid blob tool and widget handles. From left column to right column: translation handle, resize handle, and rotation handle.*

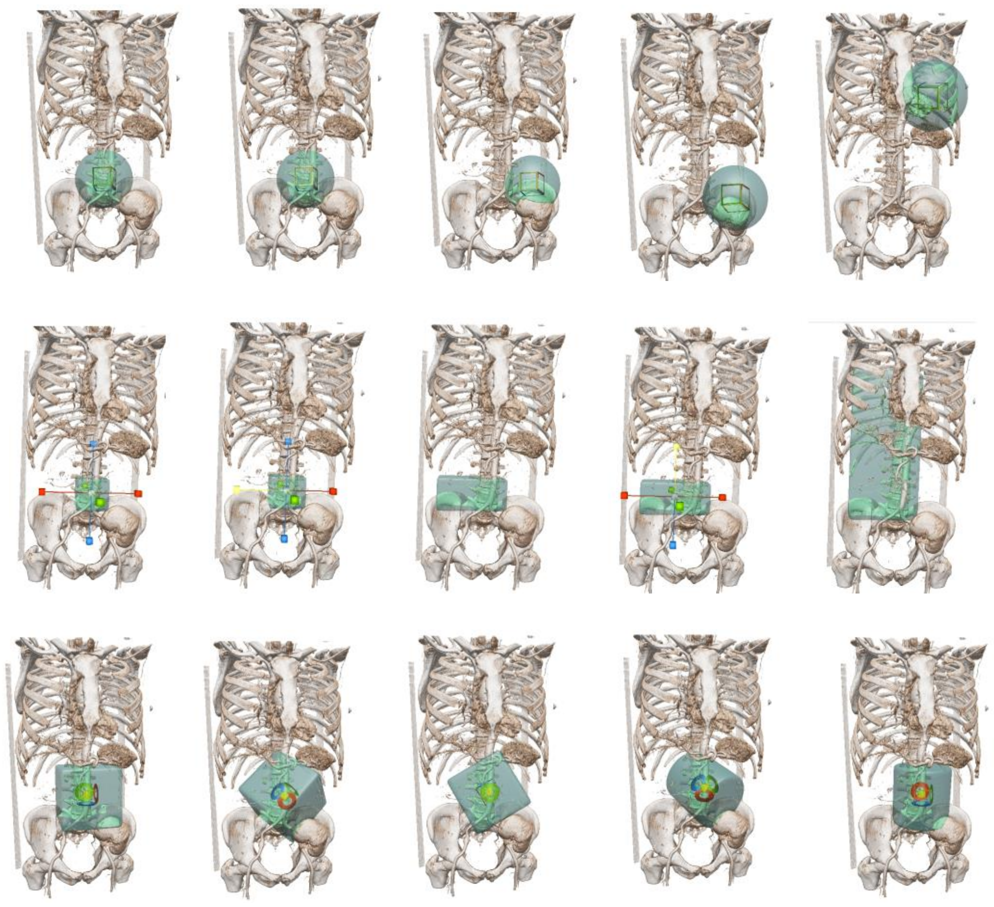


Figure 16: *Examples of blob tool interaction using the widget handles. The top row illustrates translation, the middle row resizing, and the bottom row shows blob rotation as well as two different blob shapes.*

Blob tool: As described, the blob tool is defined by a super-ellipsoid blob and has a 3D widget-based manipulation scheme (Figure 15). That is, users use the mouse to select the handles to translate, scale, and orient the blob. The handles are designed to suggest their function (i.e. translate, rotate, scale) and operation as well as provide an additional visual depth cue for the blob tool. To interact with the widget handles, a user begins by pressing and holding the shift key, causing the handles to appear near the blob. Once visible, the user selects a handle with the mouse. Once a handle is selected, holding down the left mouse button and moving the mouse allows the user to perform transformations on the blob corresponding to the current transformation mode (Figure 16). The handles are also hidden during this action to allow the user to clearly see the effects of their movements on the blob. When the user releases the left mouse button, the handles reappear. The user can also use GUI sliders to change the shape and opacity of the blob.

For precise blob positioning in any scenario or data volume, the user uses the translation widget handles. However, for quick blob positioning in a relatively noise-free CT data set, the user has the option of clicking on a point of a visible data iso-surface to instantly center the blob at the selected location. As the blob is manipulated, voxels inside are highlighted to provide visual feedback of the selection region. If a user is satisfied with the selection, they can *apply* the blob (i.e. blend it with any previously applied blobs) by pressing a keyboard shortcut or a GUI button. If the user makes a mistake they can instantly undo applied blobs with a GUI button or keyboard shortcut.

3.1.4 Super-ellipsoid Constrained Region Grow Overview

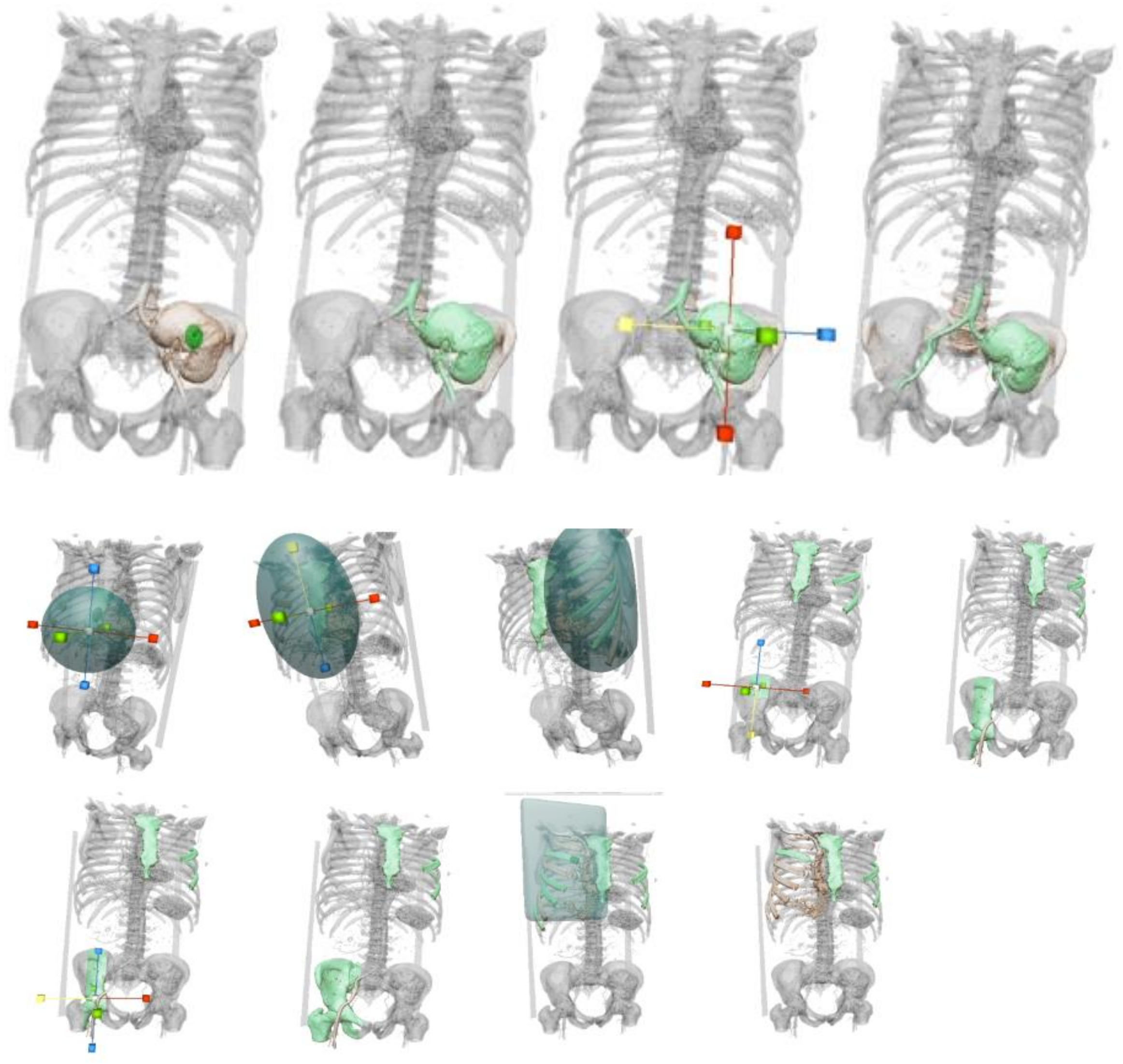


Figure 17: *Examples of how the widget-based blob region grow tool can be used to select connected structures in real-time. The region growing algorithm is spatially-constrained by the super-ellipsoid blob and the ROI can be dynamically resized in real-time by using the widget handles to adjust the blob size/shape. The ROI can also be completely removed by simply pressing a GUI undo button. Additional regions can be added by repeating the region grow algorithm at different locations in the volume.*

Highly parallel GPU-based region grow algorithms are a fast and effective segmentation technique for selecting a ROI within a volume where the ROI consists of a connected set of visible voxels. However, region grow algorithms are noise sensitive and are prone to “leaking” into neighboring regions with similar voxel intensity ranges to that of the target ROI. Therefore, interactively controlling the region grow process prevents a user from wasting time manually correcting the region grow output by removing the leaked regions. In this thesis, the same widget controlled super-ellipsoid blob UI that is used to paint an ROI is also used to interactively constrain and “steer” the region grow process. This interactive blob region grow tool can be applied directly to the volume image to select connected voxels, for example an artery “tree”, or alternatively can be applied to connected voxels within a previously “painted” ROI. In this manner, the blob tool can be combined with the blob region grow tool to allow the user to flexibly select complex shaped ROIs even within a moderately noisy volume image.

As mentioned, the constrained region grow is implemented on the GPU and the evolution of the selected region can be controlled in real time by resizing/reshaping the blob tool using the widget handles (Figure 17). The user selects the blob region grower and initiates region growing by first using the translation widget handles to place the blob around an anatomical structure they wish to select (Figure 17 upper left brown-colored region). The user then moves the mouse along the surface of this structure and a small green circular region is highlighted, indicating a valid seed voxel is within the bounds of the blob. If the circular region is colored red, this indicates the seed voxel is outside the blob boundary. A valid seed point is then selected with a mouse click. The region grow process is then activated and all valid connected voxels within the blob boundary are selected, essentially instantaneously (Figure 17, upper row, second from left). Valid voxels are voxels that are visible in the scene and not already selected. The blob can then be interactively resized, reshaped or rotated using the widget handles and the selected region will automatically grow and/or shrink, in real-time, to stay within the blob boundary (Figure 17 upper row). Other target regions can be dynamically added to the currently selected region by reinitiating the region grow tool and selecting new seed voxels (Figure 17 middle row and bottom row). If desired, the user can also use a key or GUI button to completely undo any painted region they have previously selected with the region grower.

3.1.5 Open-View

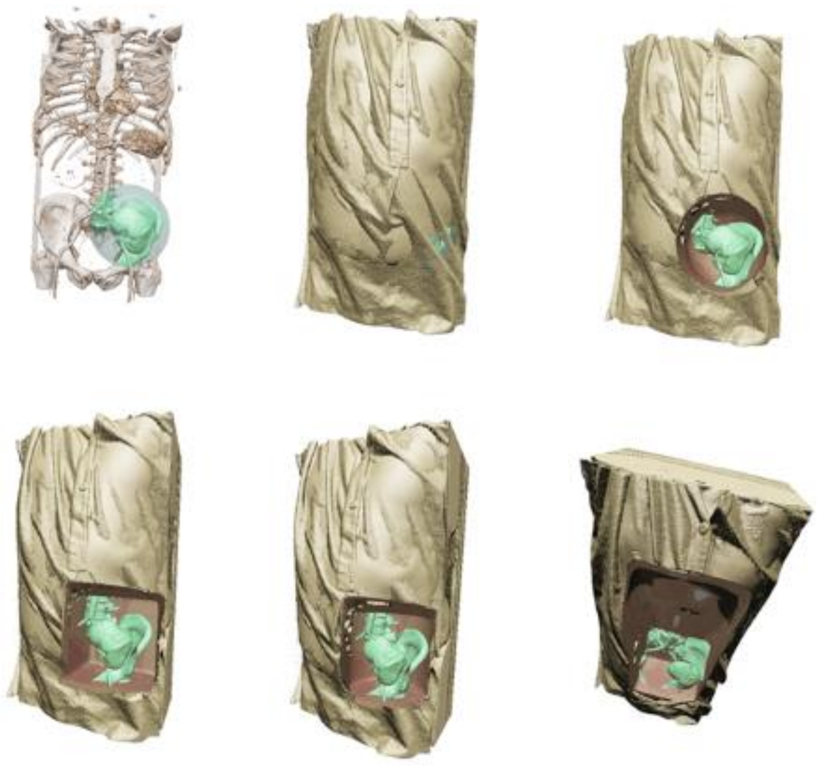


Figure 18: *Using open-view to reveal an occluded ROI. The open-view cap is defined by a super-ellipsoid blob that can be translated, scaled, rotated and shape-changed. The open-view is tied to the camera (i.e. it is view-dependent) allowing users to interactively reveal the target ROI from different viewpoints and to create cutaway surfaces of different shapes, from rounded to rectangular.*

In situations where the blob itself is occluded or the target object is occluded, the user may enable an *Open-View* mode to remove the occluding objects without losing the surrounding context. The Open-View capability is implemented with an (invisible) auxiliary "lens" with one lens endpoint fixed to the viewpoint and the other endpoint attached to the blob tool. As the user manipulates the blob tool, the Open-View lens automatically cuts away any occluding visible voxels between the viewpoint and the blob. The shape of the Open-View lens is defined by a shaft region and an adjustable super-ellipsoid cap (see Section 3.2.5 in this chapter). The cap can be translated, scaled, and rotated using a similar widget interface as the blob tool. The cap can also be tapered. GUI sliders are also available to adjust the shape of the cap – which in turn automatically adjusts the shape of the shaft to match. Combining the Open-View capability with an intuitive widget-based blob manipulation UI allows the user to quickly create specialized view dependent cutaway regions that match the geometry of the target object and the geometry of the surrounding structures such that the contextual view is enhanced with effective depth cues provided by the surface of the cutaway region (Figure 18).

3.1.6 Creating Focus plus Context Views

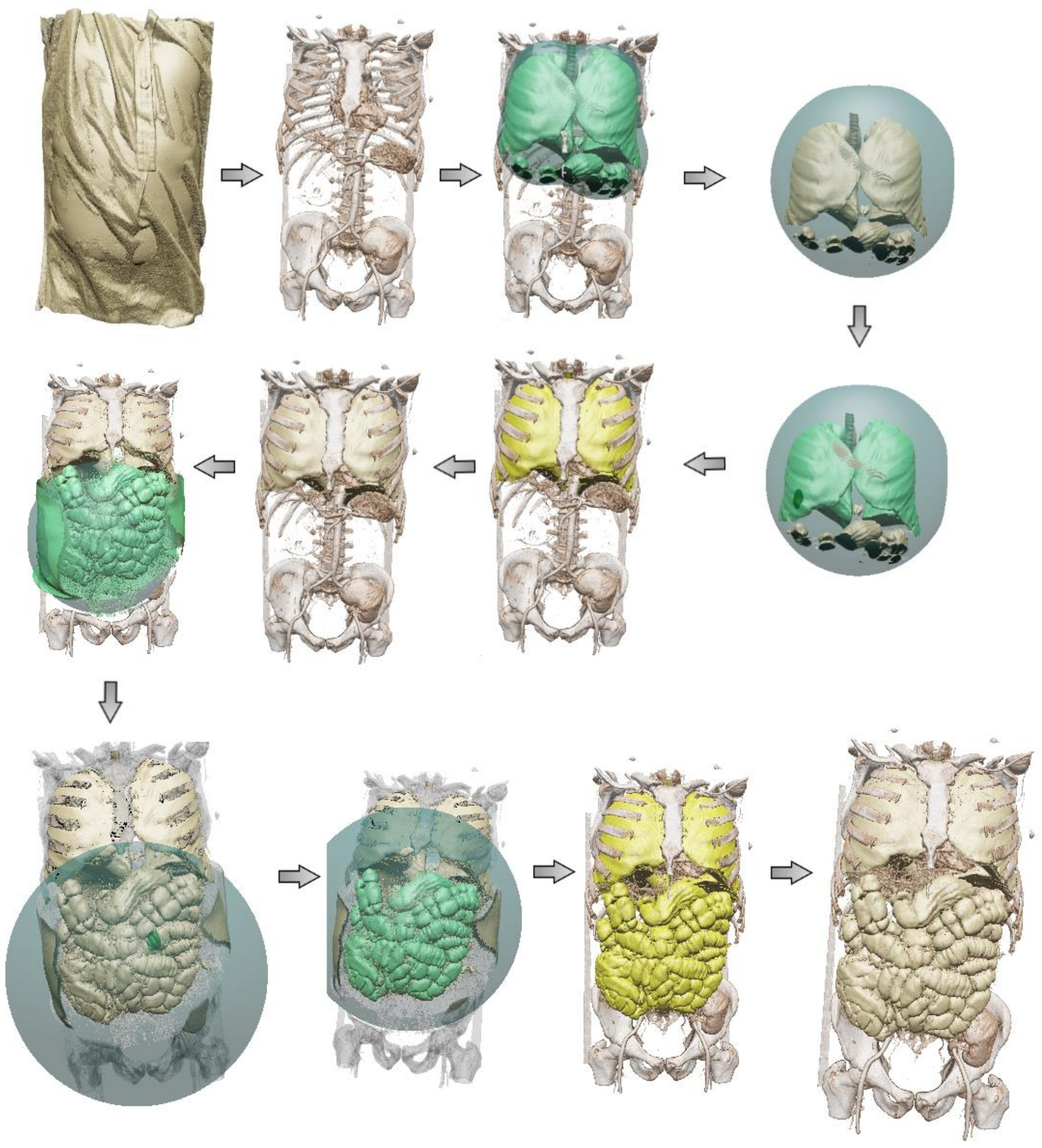


Figure 19: *Using the blob tool to paint a rough initial ROI and applying the blob region grow tool within it to select the lungs and the intestines in this CT dataset retrieved from the Osirix DICOM Image Library [13]. Voxels selected by the region growing algorithm are added to the accumulation grid and the process is repeated until the desired view is created.*

To create contextual views, many other volume rendering solutions use either multi-dimensional TFs and/or separate data-dependent segmentation techniques to produce the desired results. Unfortunately, as discussed in chapter 2, the multi-dimensional TF UI can be complex and there is no single technique that works for all situations. The system described in this thesis provides tools to break the view generation problem into a series of simpler phases using the various tools, where each tool is designed to be easy to use and the combined effect results in a desired view which can also be quickly adjusted. The system uses a data structure called the *Accumulation Grid* (see Section 3.2.6) which accumulates/combines the user's various paint actions. A disadvantage of this approach is that the effectiveness of the system is dependent on the user's ability to identify the most effective tool for each phase of the view generation task. Figure 19 is an example of how a skilled user could use the blob tool to define a ROI with different visibility settings to roughly isolate the lungs. The user can then use this ROI as the input sub-volume for the blob region grow tool to quickly select the lungs and create a contextual view that may be difficult to accomplish without a finely tuned spatial transfer function or specialized segmentation algorithm.

In the CT data set [13] (Figure 19), the arteries and bones are visible due to the minimum and maximum voxel visibility settings. Unfortunately, the lungs and intestines have a much lower intensity range and cannot be rendered with arteries and bones as the skin and clothing of the subject is occluding them. However, by first isolating the lungs and intestines and adding them to the accumulation grid, it becomes possible to have a clear view of objects of greatly varying intensities without a complicated spatial transfer function. The view can also be enhanced even further with the addition of Open-View (Figure 20). This feature allows users to create a cutaway region within the area of interest and remove occluding objects. In this case, Open-View allows the user to have a clear understanding of the lung and intestines position in relation to the subject's skin and muscles.

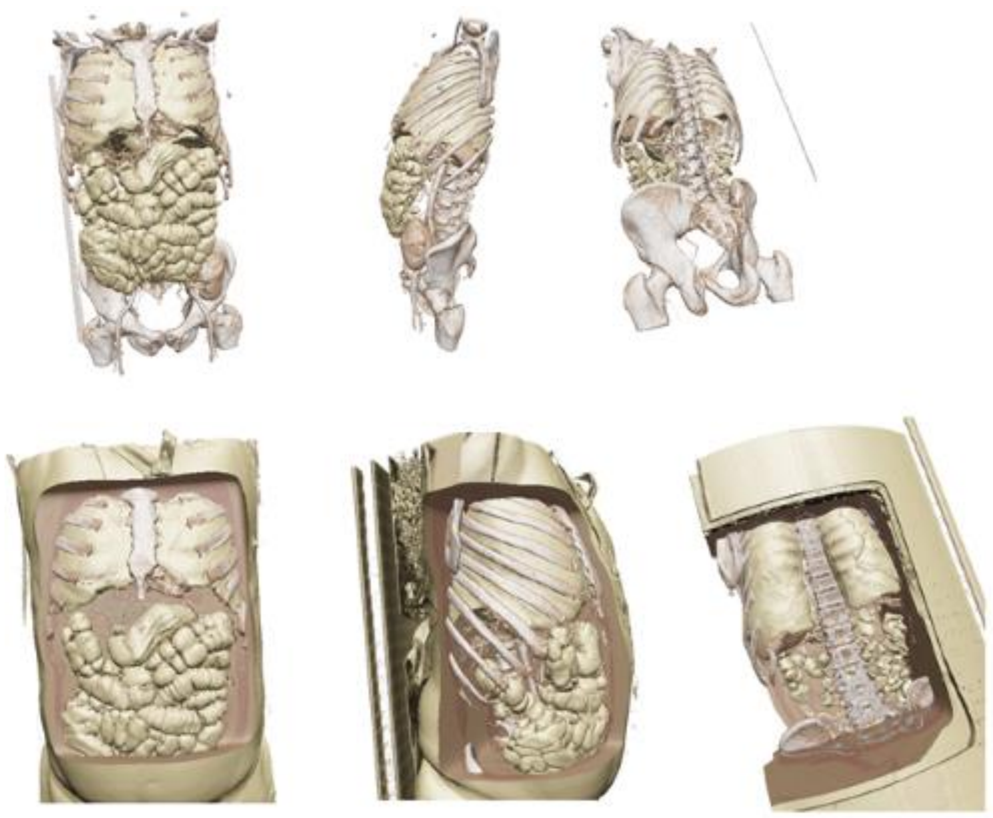


Figure 20: *Using open-view to create a specialized cutaway region to reveal a user's earlier selection of the lungs and intestines. These two structures can then be viewed in relation to the skin and muscle in the dataset.*

3.2 Implementation Details

As mentioned, super-ellipsoid blobs can be placed and blended together to define an arbitrarily shaped 3D ROI within the volume image. The super-ellipsoid blobs are defined using implicit functions and the functions are evaluated at fixed points in a special 3D grid of voxels referred to as the *blob grid*. That is, each voxel of this grid stores an implicit function field value. The blob grid has dimensions matching that of the input volume. For example, for a 256x256x256 input volume, a 256x256x256 blob grid is used. The implementation allows users to place a large number of blobs, if desired, in real time. To ensure quick lookups into the grid, the blob grid is stored in GPU memory and is modified and accessed in a highly parallel manner from GPU shader programs. Unfortunately, this implementation requires a large amount of GPU memory but results in overall higher performance by avoiding expensive calculations in the volume ray cast shader program. Specifically, this gain in performance is achieved by computing the blob grid values in a separate rendering phase using a special, highly-efficient GPU vertex shader program. As the user deposits a new blob, it is blended with the existing grid field values computed from the previously deposited blobs. There is no need to re-compute the grid field values for all blobs. Consequently, during the volume rendering phase, at each step along the ray these grid values can then be efficiently accessed from the volume rendering fragment shader program. Storing the blended blob field values in a grid is a scalable solution as additional blobs have little or no performance impact.

In addition to the blob grid, two other grids are required to support the system functionality. The dimensions of both of these grids also match the volume image dimensions. A region grow grid is used to record voxels selected by the blob region grow tool. The accumulation grid, as mentioned previously, is used to combine the voxel selections of the individual paint actions. The following sections will describe these data structures and the associated algorithms and equations used to implement the complete super-ellipsoid blob tool functionality.

3.2.1 Mathematical Formulation of Super-Ellipsoid Blending

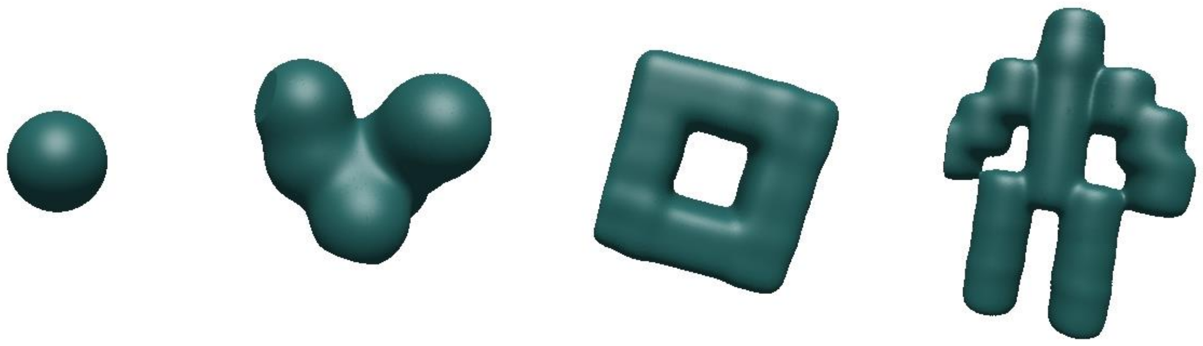


Figure 21: *Super-ellipsoid blob blending resulting in various shapes.*

A 3D region is defined in the system using a *soft object modeling* [56] approach. In the soft modeling approach, an implicit surface model is derived from a global potential field function $F(x, y, z)$ (the implicit function). This global function is defined as the sum of *n* component field functions F_i, one for each shape primitive P_i $(\text{i.e.} \, blob)$(Figure 21). That is:

$$F(x, y, z) = \sum_{i=1}^{n} F_i(x, y, z) \tag{3.1}$$

The surface of the object $\boldsymbol{S}$ may be derived from the implicit function $F(x, y, z)$ as the points in space whose value equals a threshold denoted by T (in soft object modeling *T* is often set to 0.5):

$$\boldsymbol{S} = \{(x, y, z) \in R^3, F(x, y, z) = T\} \tag{3.2}$$

Each component field function $F_i(x, y, z)$ may be conveniently defined as the composition of a distance function $d_i(x, y, z)$ and a potential function $F_i(d)$, where:

$$F_i(x, y, z) = F_i \circ d_i(x, y, z) \tag{3.3}$$

In general, function $d_i(x, y, z)$ defines the distance between a point $\boldsymbol{p}(x, y, z)$ and the center point of shape primitive P_i, while function $F_i(d)$ characterizes the blending properties of the shape primitive (i.e. how a shape primitive is blended with other shape primitives). In our implementation the blending function is defined as [57]:

$$F_i(d) = \begin{cases} 0 & , \quad d > 1.0 \\ -\frac{4}{9} * \mathrm{d}^6 + \frac{17}{9} * \mathrm{d}^4 - 22/9 * \mathrm{d}^2 + 1 , & 0 \leq d \leq 1.0 \end{cases} \tag{3.4}$$

The distance function formulation follows that of Tigges et al. [57] where $d(x, y, z)$ is a super-quadratic distance function derived from the implicit equation of a super-ellipsoid. That is, in this formulation, the super-ellipsoid shape primitive P_i is represented via the distance function. The distance function is evaluated in a normalized volume image coordinate space. The super-quadratic (sq) distance function is therefore defined as:

$$d_{sq}(x, y, z) = \left(\left| \frac{x}{r_x} \right|^{\frac{2}{\varepsilon_2}} + \left| \frac{y}{r_y} \right|^{\frac{2}{\varepsilon_2}} \right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left| \frac{z}{r_z} \right|^{\frac{2}{\varepsilon_1}} \tag{3.5}$$

The surface of a super-ellipsoid has an implicit formulation $F_{se}(x, y, z) = d_{sq}(x, y, z) - 1 = 0$ where $d_{sq}(x, y, z)$ also defines an inside-outside function. An inside-outside function provides a simple test to determine if a point $\boldsymbol{p}(x, y, z)$ is inside the super-ellipsoid ($d_{sq}(x, y, z) < 1$), outside the super-ellipsoid ($d_{sq}(x, y, z) > 1$) or on the surface of the super-ellipsoid ($d_{sq}(x, y, z) = 1$). The coefficients r_x, r_y, r_z are scale factors in the *x*, *y* and *z* directions, respectively. The parameter ε_2 controls the cross-sectional shape of the super-ellipsoid in the (x, y) plane while ε_1 controls the cross-sectional shape in a plane along the *z*-axis perpendicular to the (x, y) plane. Together these parameters control the range of influence of a super-ellipsoid shape primitive (via the distance function).

The distance function as defined in equation (3.5) above has blending problems. Tigges et al. [57] solved the blending problem by extending the distance function (for the ith shape primitive) to:

$$d_i(x, y, z) = d_{sq}(x, y, z)^{\frac{\varepsilon_2}{2}} \tag{3.6}$$

Finally, blending of the component functions F_i, is generally performed by simply summing their potential fields, as in equation (3.1). Super-elliptic blending [58] has been proposed to reduce unwanted field "bulging" effects between the component functions that can result from this simple sum. Super-elliptic blending generates a more predictable shape of the global potential field function $F(x, y, z)$. This type of blending is performed as follows:

$$F(x, y, z) = \left(\sum_{i=1}^{n} F_i(x, y, z)^K \right)^{\frac{1}{K}} \tag{3.7}$$

Where the parameter K controls the amount of blending. When $K = 1$ the super elliptic blend has the same result as the summation blend. When $K = \infty$ the super-elliptic blend is equal to the union of the component field function F_i (i.e. no blending occurs). In this thesis, K is set to a value of 8 for its balanced blending properties. See Figure 22 for an example of the effect of the parameter K.

Translating and rotating a super-ellipsoid blob is simply a matter of applying a transformation matrix **M** to the super-ellipsoid centered coordinates of equation (3.8). Representing super-ellipsoid-centered coordinates as se and volume image coordinates as vol , the transformation can be represented as:

$$\begin{bmatrix} x_{vol} \\ y_{vol} \\ z_{vol} \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x_{se} \\ y_{se} \\ z_{se} \\ 1 \end{bmatrix} \tag{3.8}$$

Where **M** is defined as:

$$\mathbf{M} = \begin{bmatrix} u_x & v_y & n_z & c_x \\ u_x & v_y & n_z & c_y \\ u_x & v_y & n_z & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

The vectors $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{n})$ are the basis vectors of the super-ellipsoid blob expressed in volume image coordinates and the point $\boldsymbol{c}(x, y, z)$ represents the blob center in volume image coordinates. That is, for a given point defined in local super-ellipsoid coordinates, transformation **M** first rotates (defined by the vectors $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{n})$) the point and then translates it by vector $\boldsymbol{c}(x, y, z)$. The transformation $\mathbf{M}^{-1}$ transforms a point in volume coordinates to super-ellipsoid coordinates. In this case, the point is first translated then rotated. Transformation $\mathbf{M}^{-1}$ can be written as:

$$\mathbf{M}^{-1} = \mathbf{RT} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

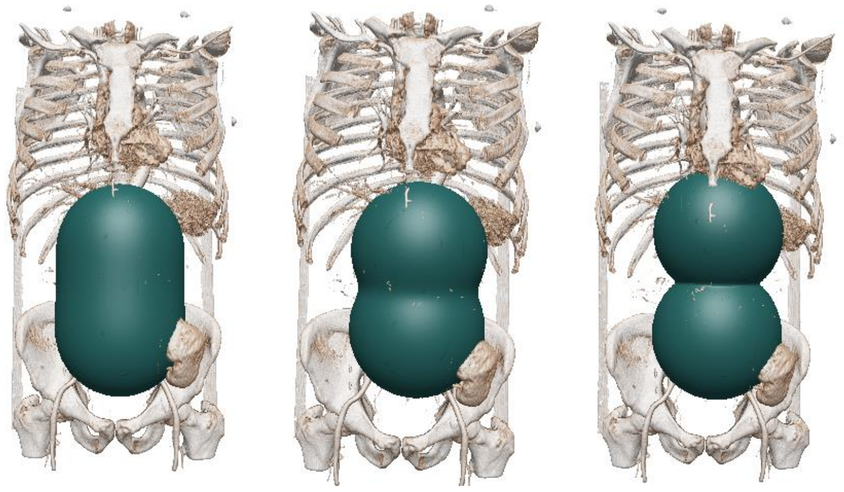


Figure 22: *Comparison of blob blending using different values for super-elliptic blending parameter K. From left to right, K= 2, 8, and 32.*

3.2.2 Super-ellipsoid Blob Grid

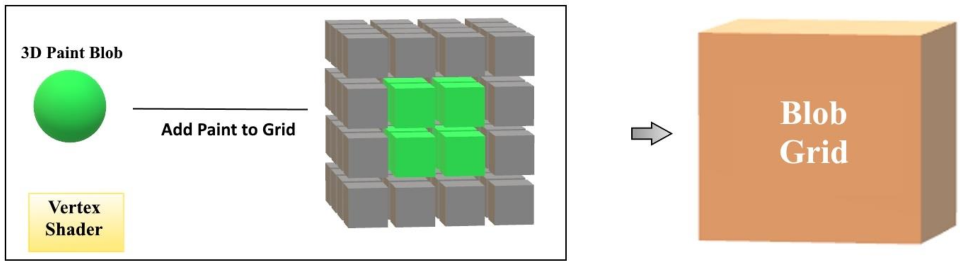


Figure 23: *A high level view of the rendering pipeline for the 3D blob grid. The blob grid is stored in GPU memory and is updated in a special GPU vertex shader program that stores blob field values at grid voxels.*

As mentioned in the introduction of Section 3.2, the system uses a special 3D grid with the same dimensions as the volume image to store blob field values. Each field value stored in the grid is updated whenever the user adds or removes a blob. However, calculating each field value sequentially on the CPU would be too computationally expensive. To ensure real-time performance, the massively parallel compute power of the GPU is leveraged by performing the blob grid calculations in a special vertex shader program (Figure 23). The blob grid is stored in GPU memory and after its field values are calculated, it can be accessed by the volume ray cast fragment shader program where volume rendering occurs. Algorithm 1 contains pseudo-code detailing how the vertex shader updates the blob field values at each grid voxel of the blob grid.

Algorithm 1 Updating Blob Grid Field Values

```
Input:
        $\boldsymbol{dim_{xyz}}$: physical dimensions of input volume
        $\boldsymbol{blobGrid}[i][j][k]$: 3D blob grid
        $\boldsymbol{c_{xyz}}$: center of input blob in volume coordinates
        $\boldsymbol{r_x, r_y, r_z}$: blob scale factors
        $\boldsymbol{R}$: blob rotation matrix from equation 3.10
        $\boldsymbol{\varepsilon_1}$: exponent from equation 3.5
        $\boldsymbol{\varepsilon_2}$: exponent from equation 3.5
        $\boldsymbol{K}$: superelliptic blend parameter
        $\boldsymbol{WyvillBasisFunc}(d)$: $-\frac{4}{9} * \mathrm{d}^6 + \frac{17}{9} * \mathrm{d}^4 - 22/9 * \mathrm{d}^2 + 1$
        $\boldsymbol{isDraw}$: $Boolean\ flag\ to\ add\ input\ blob\ field\ to\ grid$
for each grid voxel i, j, k ∈ $blobGrid[i][j][k]$ do
        $pvol_{xyz} = voxelPosition(i, j, k, dim_{xyz})$
        $pse_{xyz} \leftarrow [R] * \begin{bmatrix} pvol_{xyz} - c_{xyz} \\ 1.0 \end{bmatrix}$
        $valx \leftarrow (|pse_x|/(r_x))^{2.0/\varepsilon_2}$
        $valy \leftarrow (|pse_y|/(r_y))^{2.0/\varepsilon_2}$
        $valz \leftarrow (|pse_z|/(r_z))^{2.0/\varepsilon_1}$
        $d_{sq} \leftarrow ((valx + valy)^{\varepsilon_2/\varepsilon_1} + valz)^{\varepsilon_1/2.0}$
        **if** $d_{sq} > 1.0$ **then**
            $val \leftarrow 0$
        **else**
            $val \leftarrow WyvillBasisFunc(d_{sq})$
        **end if**
        $currentBlobValue \leftarrow blobGrid[i][j][k]$
        **if** $isDraw = true$ **then**
            $updatedValue \leftarrow (currentBlobValue^K + val^K)^{1/K}$
            $blobGrid[i][j][k] \leftarrow updatedValue$
        **else**
            $updatedValue \leftarrow (currentBlobValue^K - val^K)^{1/K}$
            $updatedValue \leftarrow \max(updatedValue ,0.0)$
            $blobGrid[i][j][k] \leftarrow updatedValue$
        **end if**
    **end for**
```

3.2.3 Region Growing Algorithm

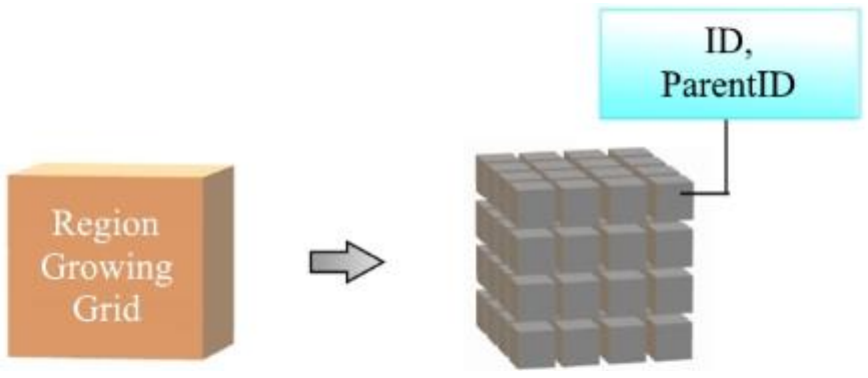


Figure 24: *Region growing grid data structure. Each grid voxel contains an ID and a ParentID. A non-zero ID indicates that the corresponding voxel in the volume image has been selected and the ParentID indicates the voxel's direction from its "parent" grid voxel.*

The GPU-based region-growing algorithm follows the algorithm described in Chen et al. [53]. The algorithm makes use of the input volume image as well as a 3D Region Growing Grid which is stored in GPU memory (Figure 24). Like the Blob Grid, the Region Growing Grid has the same dimensions as the volume image. The algorithm is divided into a vertex shader stage, geometry shader stage, and a transform feedback stage (Figure 25).

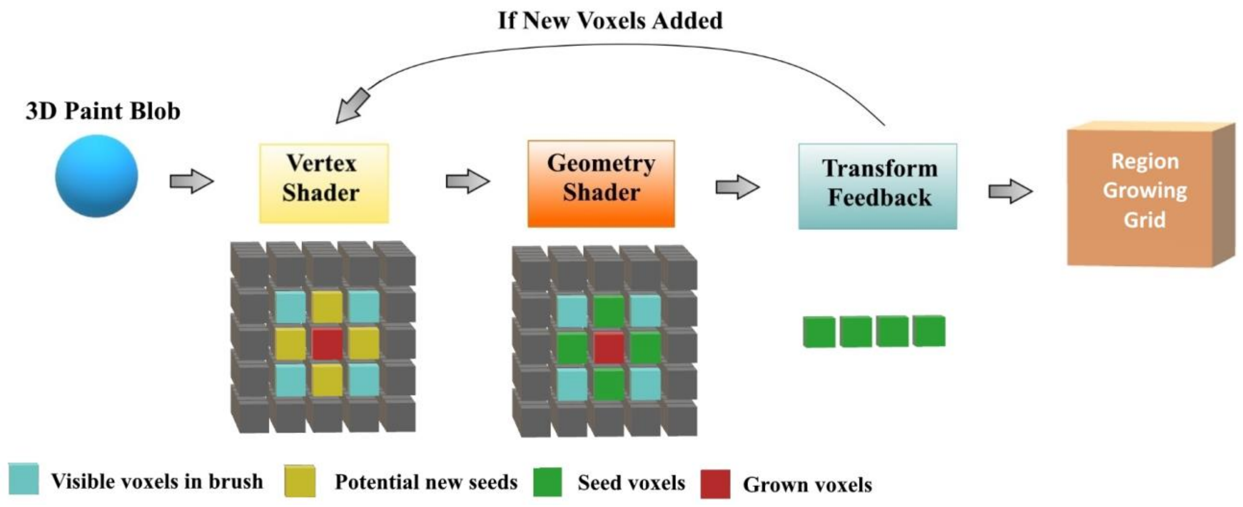


Figure 25: *GPU based region growing pipeline for updating values in the region grow grid.*

The vertex shader begins once the user has selected an initial seed voxel or when new seed voxels have been added from a previous iteration. For each new seed voxel and its neighbors, a check is made to determine whether the voxel is valid. A valid voxel is one that is visible, inside the blob, and has not already been marked as selected. If the seed voxel is valid, it is marked as selected by setting its identifier to a non-zero integer corresponding to its constraining blob’s id (red voxel). For voxels neighboring the seed, another identifier indicating the voxel’s direction from its “parent” grid voxel is stored (yellow voxel). A parent grid voxel is the voxel that (i.e. left, right, top, bottom, back, front, boundary) acted as the “parent” seed in the region grow process. For example, in Figure 25 the upper yellow voxel would have its identifier set to “top” (adjacency id of green parent voxel) while the left most yellow voxel would set its identifier to “left”. If the adjacent voxel is invalid, its parent id is marked as a boundary voxel. The adjacency information is then used in the geometry shader stage to output new seeds and boundary voxels into a special GPU memory buffer known as the *transform feedback buffer*. The data from this buffer is then passed back into the vertex shader and the process is repeated in parallel on each of these new seeds until no new seed voxels are output. Saving the seed and boundary voxels in the transform feedback buffer and using the GPU to region grow seeds in parallel allows users to dynamically shrink or grow regions in real time. Figure 26 illustrates this process with a 2D example and shows how values are set in the region grow grid for each new seed point passed to the region grow shaders.

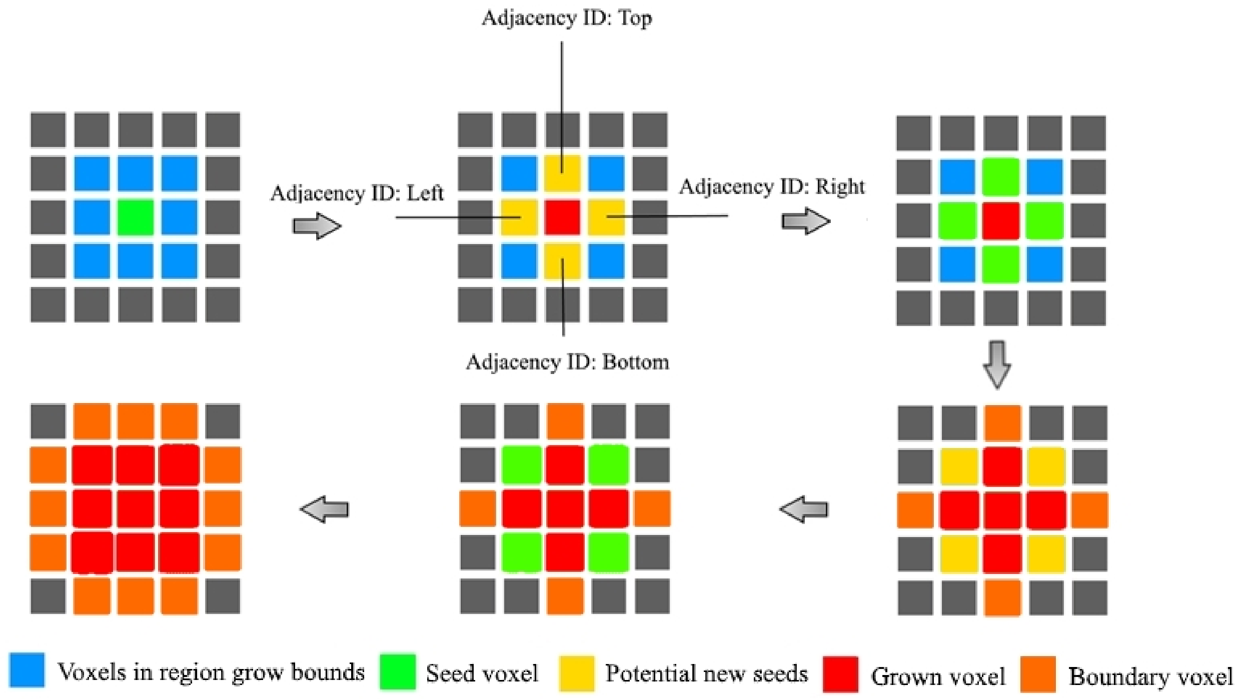


Figure 26: *A 2D representation of the region grow algorithm. Starting from the top left: 1. Initial seed voxel. 2. Initial seed marked as grown and valid adjacent voxels marked as potential new seeds. 3. Potential new seeds changed to new seeds by geometry shader. 4. Vertex shader marks new seeds as grown and flags adjacent voxels. 5. Potential new seeds changed to new seeds by geometry shader. 6. No new seeds, region grow complete.*

For simplicity, Figure 26 only illustrates a 2D example with a maximum of 4 seeds. However, a real scenario may have thousands of seeds processed in parallel and may result in duplicate neighbor voxel output. To reduce redundancy and ensure the algorithm runs at interactive rates, the adjacency information is used by the geometry shader to determine which grown voxel is responsible for emitting new seeds [53]. Figure 27 illustrates how the geometry shader uses adjacency information to eliminate duplicate output.

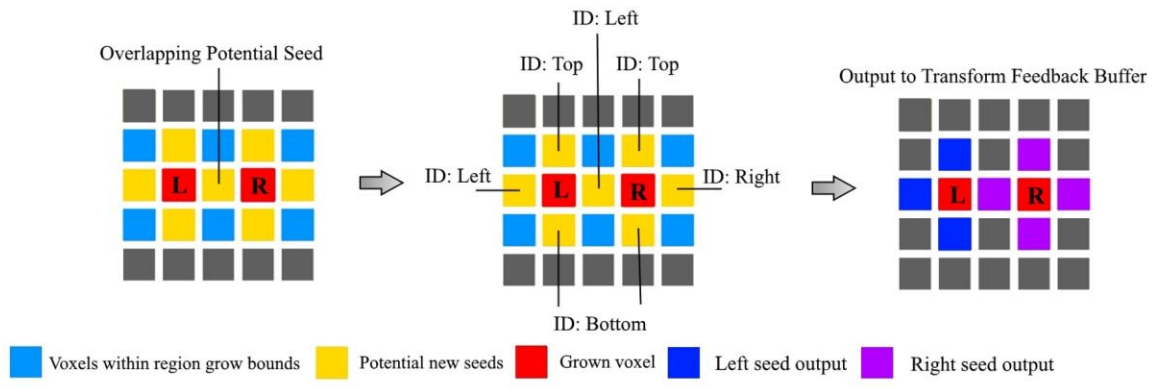


Figure 27: *A 2D representation of geometry shader region grow algorithm. Potential seeds are flagged in parallel in the vertex shader and adjacency information is used by the geometry shader to remove redundant output.*

When the region grow bounds defined by the blob region grow tool are changed, (i.e. the user changes the size and/or shape of the blob) the algorithm uses the boundary voxels stored in the transform feedback buffer as new seeds. The algorithm then performs a region grow or region shrink depending on the new dimensions of the region grow bounds. In the case where a boundary voxel isn't adjacent to a valid voxel, the algorithm performs a region shrink. The shrink first checks to see if the current boundary voxels are still valid by checking if there are any grown adjacent voxels within the region grow bounds. If all adjacent voxels are invalid, the algorithm performs a region shrink by setting the current voxel as empty and flagging all neighboring non-empty voxels as new seeds. The new seeds are then stored into the transform feedback buffer and the process is repeated until no new seeds are added. Figure 28 illustrates a 2D example of how the algorithm performs a region grow shrink.

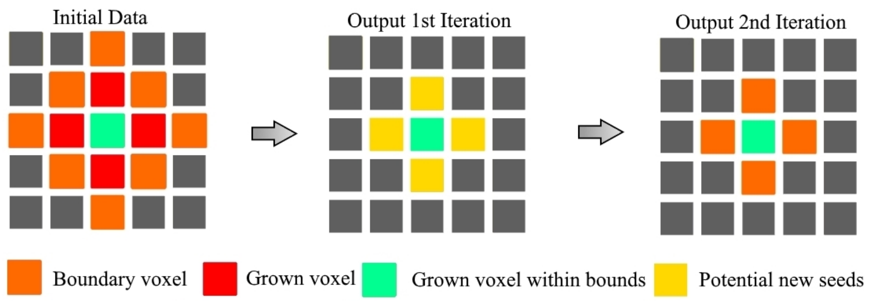


***Figure 28**: A 2D representation of a region shrink over two iterations. Starting from left to right: 1. Initial region shrink scene. 2. Boundary voxels removed and grown voxels marked as potential new seeds 3. New seeds marked as boundary voxels.*

3.2.4 Open View

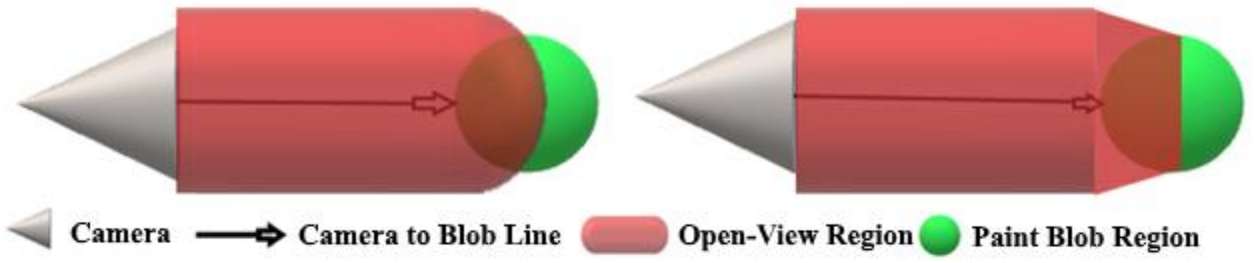


***Figure 29:** An open view representation for spherical and tapered cubical cap shapes.*

As described in Section 3.1.4, the open-view capability is implemented with an (invisible) auxiliary "lens", with one lens endpoint fixed to the viewpoint and the other endpoint attach to the blob tool. The cap of the Open-View lens (Figure 29) is defined by one half of a super-ellipsoid, optionally tapered, and allows users to create open-view lenses of varying shapes. The region created by the Open-View lens is determined by a line starting from the camera to the center of the blob tool. During volume ray casting, a calculation is made to determine the coordinate on the line that produces the shortest distance between the current ray position and the Open-View line. This coordinate is then used as the origin of a super-ellipse with the same shape parameters as the cap. The series of super-ellipses along the line constitutes the shaft of the Open-View lens and all voxels within this area and not within the blob are hidden from view. If the Open-View cap shape is spherical, it results in a cylindrical tube shaft with a (hemi-)spherical cap at the end. For cubical shapes, the user has the option of tapering the end of the cap. A spherical cap or a tapered cubical cap result in cut surfaces of objects surrounding the target ROI (i.e. inside the blob) that are oriented toward the viewer, regardless of how deep the target ROI is within the volume. Based on experiments, this shaft-plus-cap Open-View design results in a superior depth cue for the occluded target ROI than using a simple frustum shaped Open-View or a single super-ellipsoid shaped Open-View. In addition, the flexibility of the adjustable super-ellipsoid cap shape provides greater cutaway shape control.

3.2.5 Accumulation Grid

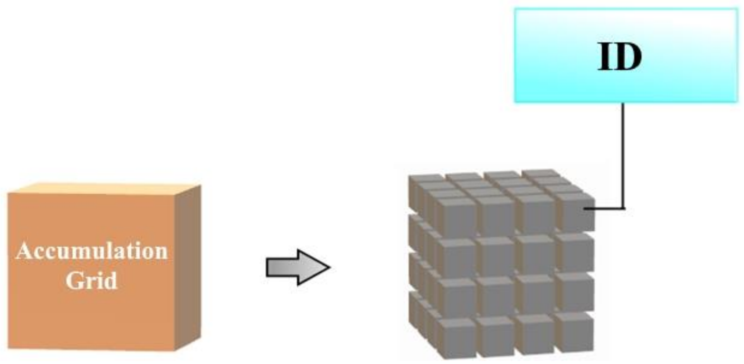


Figure 30: *A representation of the accumulation grid. Each voxel in the 3D grid contains an id and matches the dimensions of the volume data.*

The purpose of the accumulation grid is to give users the ability to combine ROIs of different visibility settings. The grid is stored in GPU memory and is a data structure matching the dimensions of the input volume (Figure 30). Each index in the grid holds an id that represents an index into an array of transfer function parameters that define voxel visibility. Currently, the transfer function parameters consist simply of two values representing the minimum and maximum voxel intensity visibility range. Future implementations can be extended to define more complex voxel visibility rules.

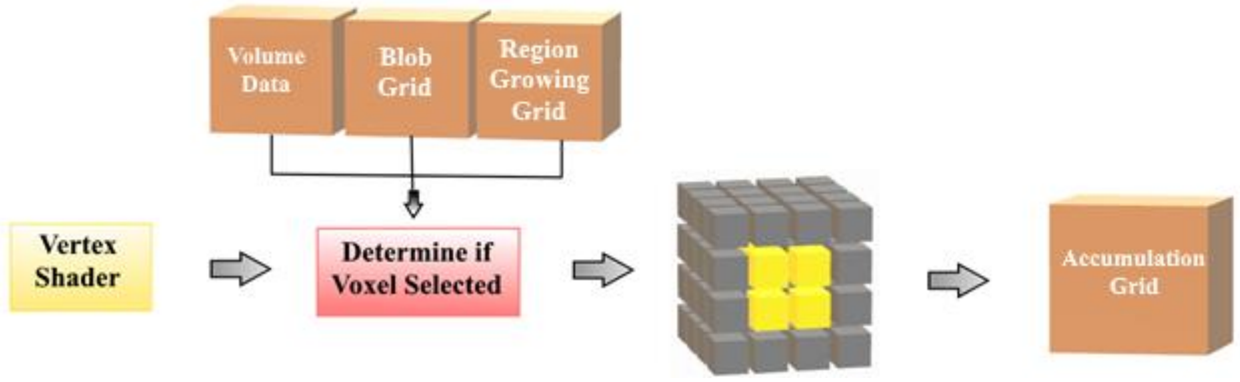


Figure 31: *GPU pipeline for updating the accumulation grid. Voxels selected in the blob grid, or region growing grid are added to the accumulation grid by a special vertex shader.*

The process of adding data to the accumulation grid is handled in parallel by a special vertex shader (Figure 31). This vertex shader evaluates each voxel in the grid against the values stored in the blob grid or region growing grid. Voxels marked as selected in the blob grid or region growing grid are assigned an ID in the accumulation grid. An ID of zero represents an empty value and an ID greater than zero represents a voxel that has been added to the accumulation grid. If a voxel in the accumulation grid already has an ID greater than zero, the incoming data is discarded. This was done to prevent users from overriding previously selected areas unless explicitly requested by the eraser function. Figure 32 illustrates how a user can add selected voxels to the accumulation grid from either the blob grid or region growing grid.

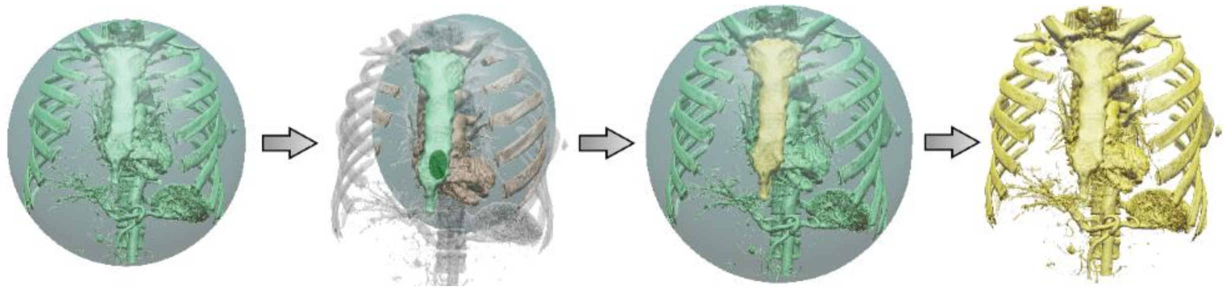


Figure 32: *An example of how the blob region grow tool can be applied to a region defined by the blob tool and added to the accumulation grid. From left to right: 1. Blob tool used to create an ROI of the sternum and ribs. 2. Blob region grow applied to sternum. 3. Region grow paint or blob paint can be added to accumulation grid.*

3.2.6 Rendering Pipeline

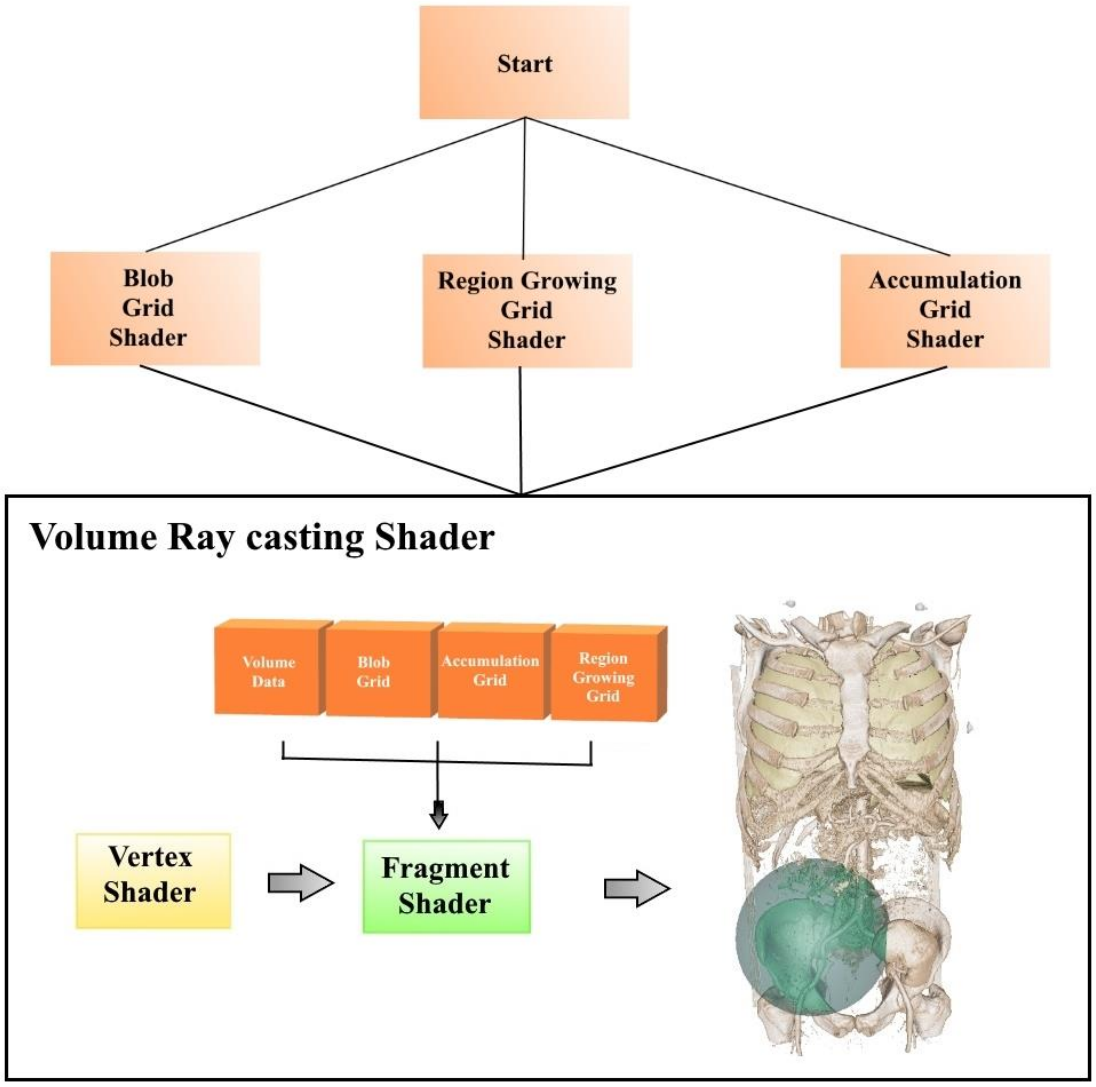


Figure 33: *Rendering pipeline for the volume ray cast shader. The vertex shader defines the bounds of the volume and the fragment shader accesses voxels from the input volume, blob grid, region growing grid, and accumulation grid to render the scene.*

Figure 33 illustrates the rendering pipeline of our system and begins with a check to determine if the blob grid, accumulation grid, or region growing grid needs to be updated. In the case where an update is required, the corresponding shader is executed. Once the grids are updated, they are passed into the volume ray casting shader to render the final scene. The volume ray casting shader is divided into a vertex shader and fragment shader with most of the work done in the fragment shader. The vertex shader creates the boundaries of the volume ray caster and the fragment shader assigns a color to each screen pixel by stepping along each ray in parallel. For each step along the ray, the algorithm uses the volume data, blob grid data, region growing grid data, accumulation grid data, and the separate minimum/maximum voxel visibility setting for the context region and the painted blob region(s) to determine the final rendering.

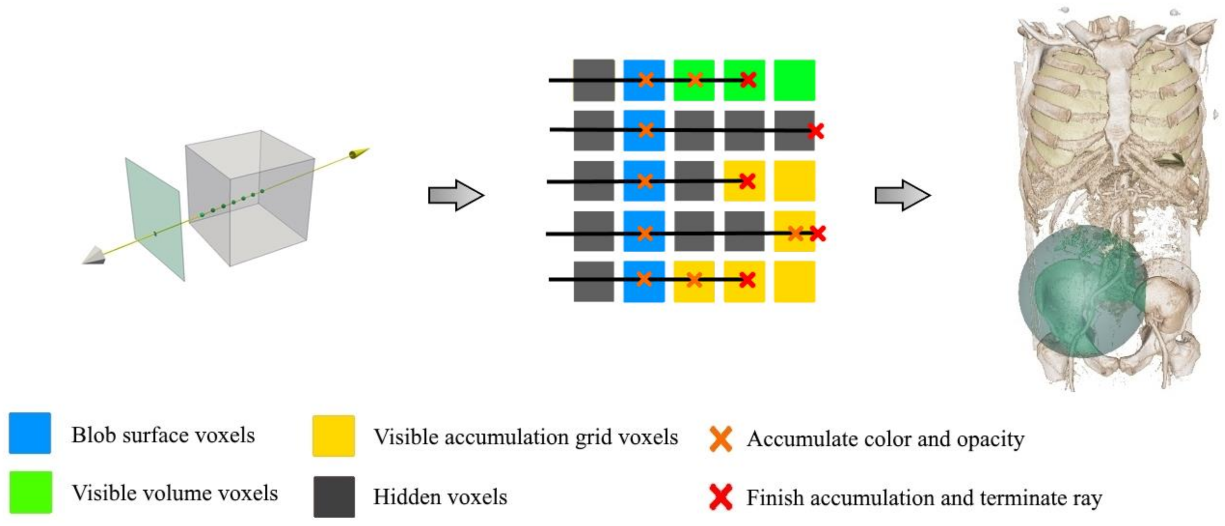

Blob surface voxels
Visible volume voxels
Visible accumulation grid voxels
Hidden voxels
Accumulate color and opacity
Finish accumulation and terminate ray

Figure 34: *A 2D and 3D representation of color accumulation in the fragment shader. Rays are cast from the camera into the volume. As the ray steps through the volume, each visible voxel's color and opacity are accumulated and blended until an opacity value of 1.0 or greater is reached or until the ray hits the bounds of the volume. From left to right: 1. A 3D visualization of volume ray casting, where each dot represents a step along the ray. 2. A 2D visualization of volume ray casting, where each box represents a ray's color/opacity accumulation at each step. 3. The result of rendering the color and opacity accumulation for each ray.*

Figure 34 illustrates how the volume ray caster steps along a ray and assigns a color to a pixel. Each ray starts with an initial entry coordinate and points along the ray are sampled until an opacity of 1.0 is reached or the bounds of the volume are hit. At each step along the ray, a blob field value is calculated and if the value is greater than 0.5 the blob surface is rendered. The shader then checks if the current voxel selected from the volume data is visible by comparing its intensity against the data from the blob grid, accumulation grid, and open view. If the voxel is visible, its color and opacity are accumulated into the result.

The blob surface of a ROI is rendered in the volume ray caster using the previously computed field values in the blob grid. For the preview blob, the blob field is calculated inside the volume ray caster and can be dynamically resized, translated, rotated and blended with the blob grid data in real time. Voxel visibility is determined by the accumulation grid, blob grid, open view, and the minimum and maximum visible voxel intensity ranges. For overlapping regions, there is a visibility hierarchy and the order is as follows: accumulation grid voxels, preview blob voxels, open view voxels, blob grid voxels, and volume data voxels. For example, a voxel inside the accumulation grid that is marked as invisible will not be shown even if it is within the minimum and maximum visible voxel intensity range for the blob or volume regions (Figure 35).

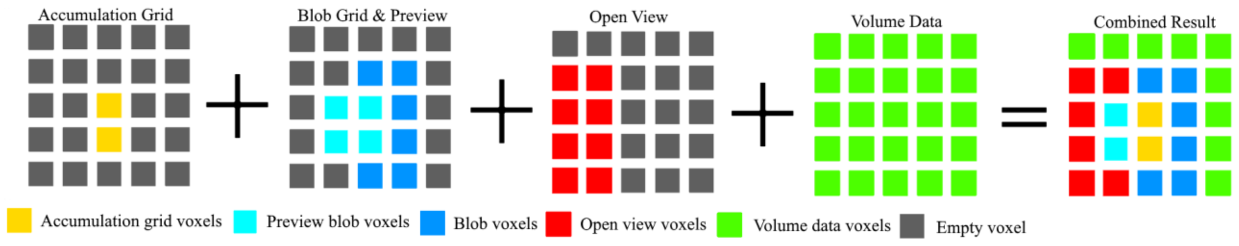


Figure 35: *A 2D representation of the voxel visibility hierarchy for overlapping regions.*

Chapter 4: User Study and Experimental Results

In this chapter we present the results of a user study that evaluates the effectiveness of our widget-based super-ellipsoid painting and region growing system for exploring and selecting areas of interest within a 3D volume. The user study compares the blob tool to a surface brush technique and a screen brush technique. These tools are described in Section 4.1.3. Also, in this chapter, several experiments are presented illustrating the contextual view generation capabilities of the system. The data sets used in the experiments are downloaded from the Osirix DICOM Image Library [13]. All the results were generated on a Windows 10 desktop computer equipped with a Nvidia 1080 GTX graphic card on a 1920x1080 native resolution monitor.

4.1 User Study Description

The user study was divided into four parts. Each part required participants to select a highlighted target object. The goal was to compare the widget-based UI selection tools (the blob tool and the blob region grow tool) to other types of tools with well-known UIs according to time, accuracy, and user preference.

4.1.1 Participants

Sixteen people participated in the study. Each participant took an average of one hour to complete the study. The participants were asked to paint several regions of interest with four different paint-based selection tools and then fill out a questionnaire. Participants consisted of 14 males and 2 females aged between 18-30 years old. Participant's average mouse usage per week was approximately 28 hours, with an average of 11 hours a week for video games and 2.4

hours for 3D modeling software. All 16 participants had a computer science or engineering background and didn't have any medical experience.

4.1.2 Apparatus

The user study experiments were performed on a Windows 10 desktop computer with a mouse and keyboard. The computer was equipped with Nvidia 1080 GTX graphic card to ensure the system ran at a smooth 60 fps on a 1920x1080 native resolution monitor.

4.1.3 Techniques

The four tools used in the study were the blob tool and widget-based blob region grow tool, a surface brush and a screen brush. The UI of the surface brush and UI of the screen brush are described below.

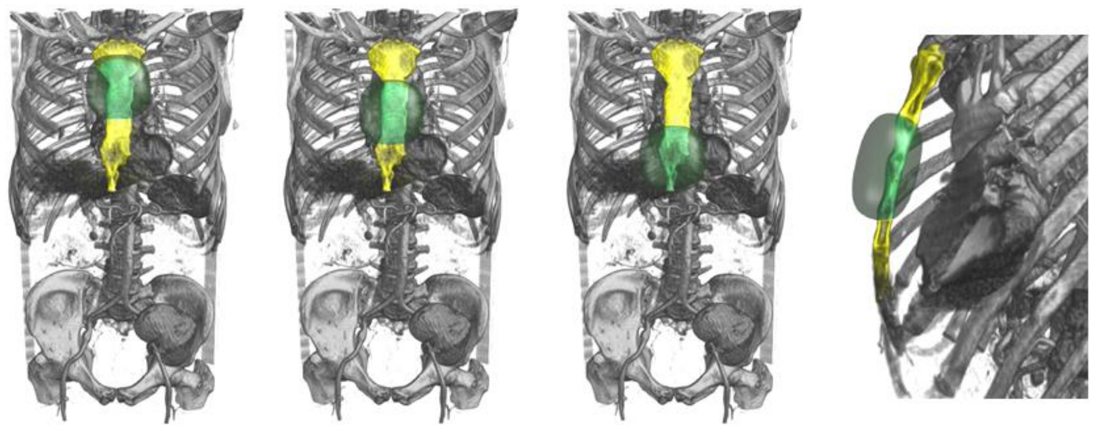


Figure 36: *Illustration of the surface brush UI. The user moves the mouse and the brush blob automatically slides along the data iso-surface under the cursor. The blob also orients itself to line up with the data surface normal. The rightmost figure shows a side view and illustrates this surface normal alignment.*

Surface brush: The surface brush (Figure 36) was implemented in a separate system [5]. However, the UI and volume rendering parameters of this system were updated to match those of the blob tool, screen brush and blob region grow tool system. The surface brush is defined by a super-ellipsoid blob similar to the blob tool, but instead of widget handles, the surface brush has an adjacent GUI panel with sliders and buttons to control brush size, orientation, and shape. The surface brush also had two painting modes, painting-plane sliding mode and surface sliding mode. In painting-plane sliding mode, the user presses and holds the left mouse button. As the user moves the mouse, the brush automatically adheres to and slides along the surface of an invisible oriented painting plane. The orientation of the painting plane is automatically tied to the camera view plane orientation. The depth of the painting plane in the volume image (i.e. distance from the camera) can be controlled with the mouse scroll wheel that translates it along the view plane normal. Surface sliding mode is the primary painting mode and allows the user to slide the brush blob along data iso-surfaces in the volume rendering by moving the mouse cursor (while pressing and holding the left mouse button) (Figure 36). In addition, if the mouse is moved to a new iso-surface position and the left mouse button is clicked, the brush will automatically jump to that surface position. Paint mode switching is handled by a button in the GUI panel. Similar to the blob tool, voxels encompassed by the paintbrush blob are highlighted to indicate the effect a paint action would have on the view. If a user is satisfied with the brush position, they can deposit the paint brush blob with a key press or with a GUI button. If the user makes a mistake, they can undo a deposited blob with an undo key or GUI button.

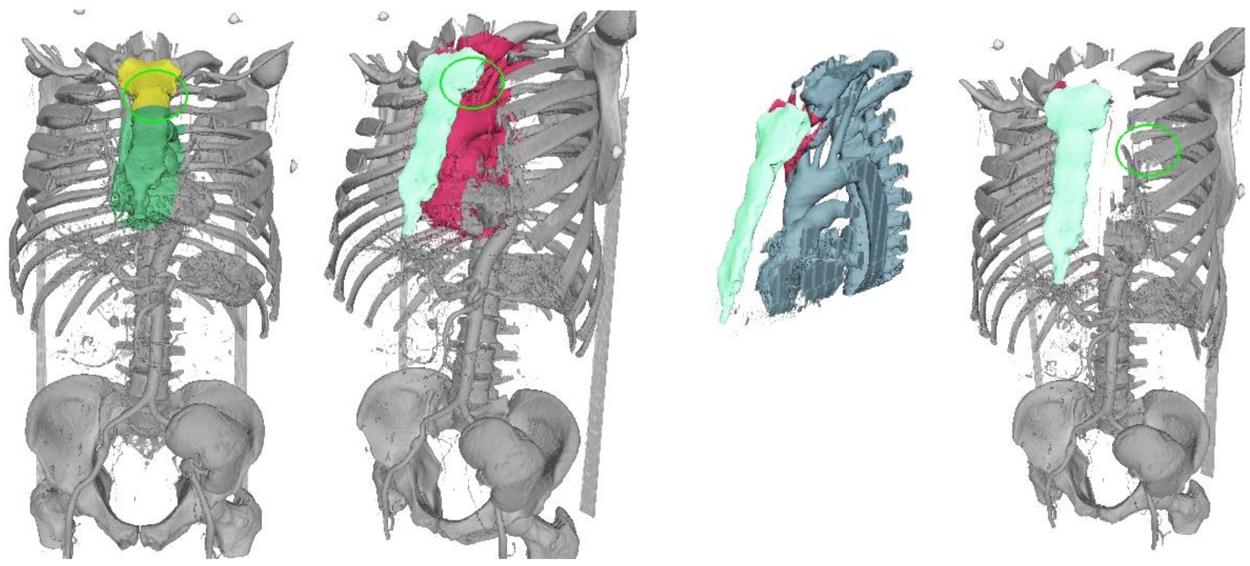


Figure 37: *Illustration of the screen brush. The brush has a circular outline and users can paint strokes by holding down the left mouse button and dragging the cursor along the screen. From left to right: 1. Painting the sternum until all target voxels (yellow) are selected (green). 2. Rotating the view to reveal the additional voxels selected in depth. 3. Switching the brush to erase mode to erase unwanted selections. 4. Apply erase and inspect the result.*

Screen brush: The screen brush is a screen space painting technique similar to the ones used in many other 2D painting applications. Centered around the mouse-controlled cursor, is a circular outline indicating the bounds of the screen brush which can be resized using a separate GUI slider. Users can paint on a view by holding down the left mouse button and dragging the mouse cursor along the screen to paint strokes (Figure 37). Since the view is defined in a 3D space, the brush also paints in depth and selects voxels based on the camera's perspective projection. This results in a brush that appears to be painting in a 2D view but is painting a 3D cone that gets wider based on the distance from camera. The depth of the cone extends to the far side of the volume image. Therefore, to select a target object, users must rotate the view with the mouse to find a screen view such that the target object is visually separated from the surrounding objects.

After painting the object from this viewpoint, the users must then rotate the view again to a new viewpoint. They then paint any missed portions of the target and/or erase painted regions on neighboring objects that were behind the target object when painting from the previous viewpoint. This process is repeated until the target object has been selected. Voxels underneath the screen brush are highlighted to indicate the effect a paint action would have on the view. If the user is satisfied with the highlighted region, they can press a GUI button to add the paint to the accumulation grid or press a key or GUI button to undo the paint. Once paint is added to the accumulation grid users can switch the brush to erase mode and erase incorrectly selected areas with the same technique. Although this painting style requires a user to continually rotate the entire view and painting/erasing to refine the selection, the painting action itself is familiar and intuitive to users.

4.1.4 Datasets

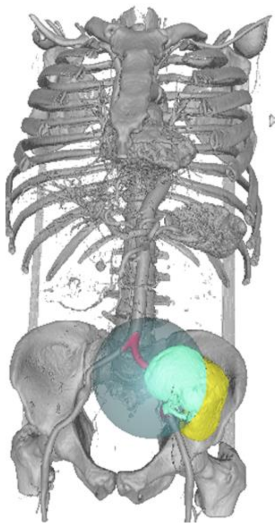


Figure 38: *Illustration of the user study selection task. The yellow highlighted voxels are the target region (in this example, a kidney transplant). If the user positions a paint tool such that these yellow voxels are inside the bounds, they are highlighted green. If the paint tool encompasses non-target voxels, they are highlighted red.*

The study used two CT scans [13] and was comprised of four different target anatomical structures. Each structure was specifically selected to tease out the strengths and weaknesses of each painting UI. In the hands of an experienced user, the selection tools used in the study can be used to quickly isolate a complex-shaped ROI consisting of multiple anatomical structures. However, in this study the participants are naïve users. Consequently, an ROI consisting of single, clearly defined anatomical structures were chosen as the targets. The target structures were highlighted in yellow and the remaining visible voxels were de-emphasized by coloring them gray (Figure 38). If the user correctly selects these voxels, they are instantly painted green. Incorrectly selected voxels (i.e. voxels inside the paint bounds but outside the target) were instantly painted red.

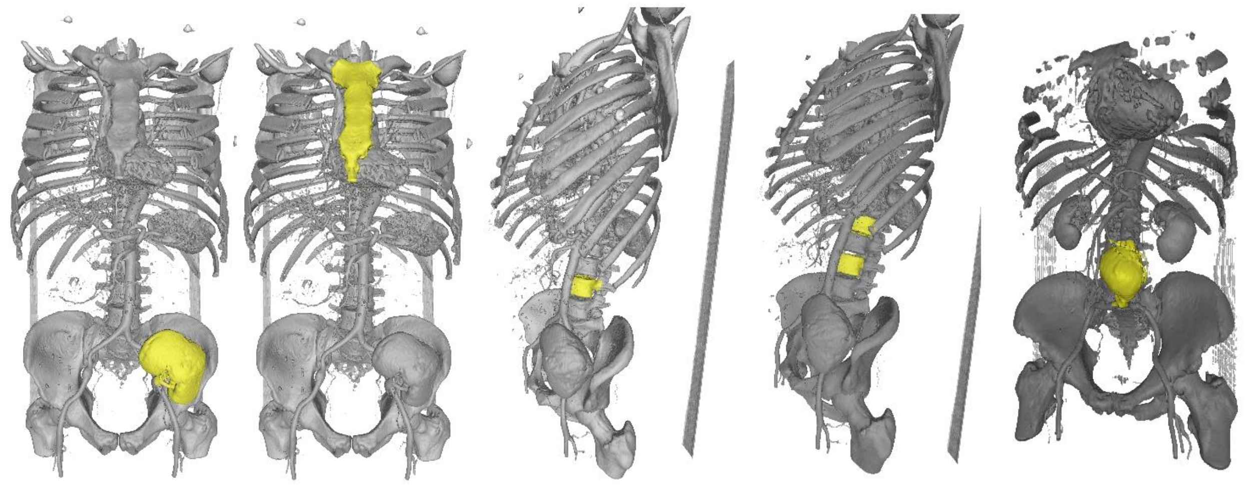


Figure 39: *A series of anatomical structures used in the user study. From left to right: kidney, sternum, single vertebra, double vertebra and aneurysm.*

Kidney transplant: The kidney transplant structure (Figure 39, left) was chosen because it was a challenging structure to isolate. Participants were tasked with painting the kidney while minimizing the amount of “spilled” paint (colored red) on surrounding structures such as the surrounding arteries and hip bone.

Sternum: The sternum structure (Figure 39, second from left) was chosen because it was an easy target to isolate accurately. The goal was to determine which technique had the best results when selecting a relatively clean object.

Vertebra: A vertebra (Figure 39, middle) was chosen because it was a challenging structure to isolate and typically requires resizing interactions. The goal was to evaluate the effectiveness of the widget based resize handles against the GUI slider-based resizing of the surface brush and screen space brush.

Double Vertebra: The double vertebra target (Figure 39, second from right), as the name implies, adds another vertebra in the spine and is used for the region grow part of the user study. This structure was the most difficult to select.

Aneurysm: A CT scan containing an aneurysm (Figure 39, right) was used in part II of the study that tested the widget-based blob region grow tool. Specifically, it was used to test the effectiveness of the blob region grow tool’s widget interface compared to an unconstrained region-grow brush with simple mouse control for growing and shrinking the selected region.

4.1.5 Accuracy Measures

We follow Yu et al. [59] and use the recent American Psychological Association recommendations [60] that report results using estimation techniques and confidence intervals. To compare the four techniques, we used task completion times and an F1 accuracy measure to evaluate the accuracy of our results. An F1 measure is calculated from the number of true positives (TP, correctly selected voxels), false positives (FP, incorrectly selected voxels), and false negatives (FN, correct voxels not selected). F1 is calculated as $F1 = 2 * (P \cdot R)/(P + R)$ with precision $P = TP/(TP + FP)$ and recall $R = TP/(TP + FN)$. An F1 score of 1 indicates a perfect performance, while a score of 0 represents the worst possible result. For accuracy, a 95% confidence interval was calculated.

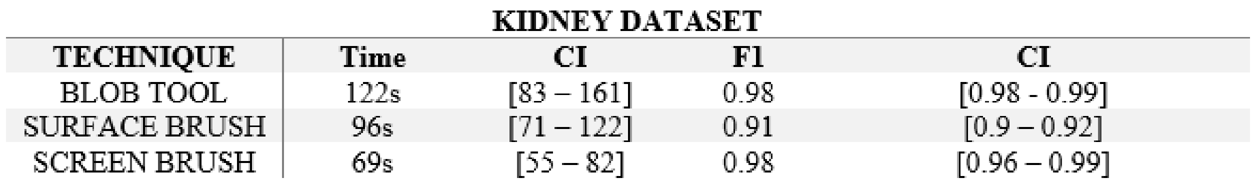
4.2 User Study Results

This section will provide an analysis of the user study results, results of the questionnaire, and observations from watching participants complete the study.

4.2.1 Part I: Blob Tool vs Surface Brush vs Screen Brush

The first part of the study was designed to compare the blob tool with the surface brush and screen brush. These three paint tools are all interactive geometric selection techniques that can be used to spatially isolate a 3D ROI. Participants were asked to select all voxels of the target object as quickly and accurately as possible with target voxels highlighted in yellow. If non-target voxels were selected, they were highlighted in red, while correctly selected voxels were highlighted in green. Since participants were given a visual cue when voxels were incorrectly selected, they were told that they could have a little bit of spilled paint if they had made a reasonable attempt. This was mentioned to prevent participants from redoing their paint until they had made a perfect selection. Giving participants visual feedback also mitigates the advantage a user with knowledge of anatomy and medical images may have over a naive user. Participants were also shown the open-view functionality and demonstrations were given for each technique. They were also given a few minutes to practice with each interface before beginning the trials. The surface brush, blob tool, and screen brush were tested on the kidney transplant, sternum, and vertebra datasets. The blob region grow tool was tested on the kidney transplant, aneurysm, and double vertebra. Two trials were performed for each target object. The first trial for each dataset was used for practice and its results were discarded. However, participants were not told this information and performed both trials as quickly and accurately as possible. The order of the techniques the participants used followed a round robin approach except for the region grow technique always being last. For example, the first participant would conduct trials in the order of screen brush, surface brush, blob tool and blob region grow tool. The second participant would be blob tool, screen brush, surface brush, and blob region grow tool. The third participant would be surface brush, blob tool, screen brush and blob region grow tool, and so on. This round robin approach to testing was done to insure no selection technique had a learning advantage over other. The region grow technique was exempt from this because it was a semi-automatic segmentation technique that used the blob tool interface and we didn't want it to influence participants performance and answers on the questionnaire with respect to the purely geometric selection techniques. Participants were also allowed to undo/redo any of the selections during a trial and once they felt they had completed their task, they could press a finish button to begin the next trial. Once participants had completed the trials for the screen brush, surface brush, and blob tool they were given a questionnaire to evaluate their level of satisfaction for each tool they had used. Once they had finished evaluating the three tools they were asked to perform the blob region grow tool trials and finish the questionnaire.

4.2.1.1 Kidney Results



KIDNEY DATASET				
TECHNIQUE	Time	CI	F1	CI
BLOB TOOL	122s	[83 – 161]	0.98	[0.98 - 0.99]
SURFACE BRUSH	96s	[71 – 122]	0.91	[0.9 – 0.92]
SCREEN BRUSH	69s	[55 – 82]	0.98	[0.96 – 0.99]

Table 4.1: *Numerical values of the average task completion time (in seconds) for each selection technique on the kidney dataset. Error calculated with a 95% confidence interval.*

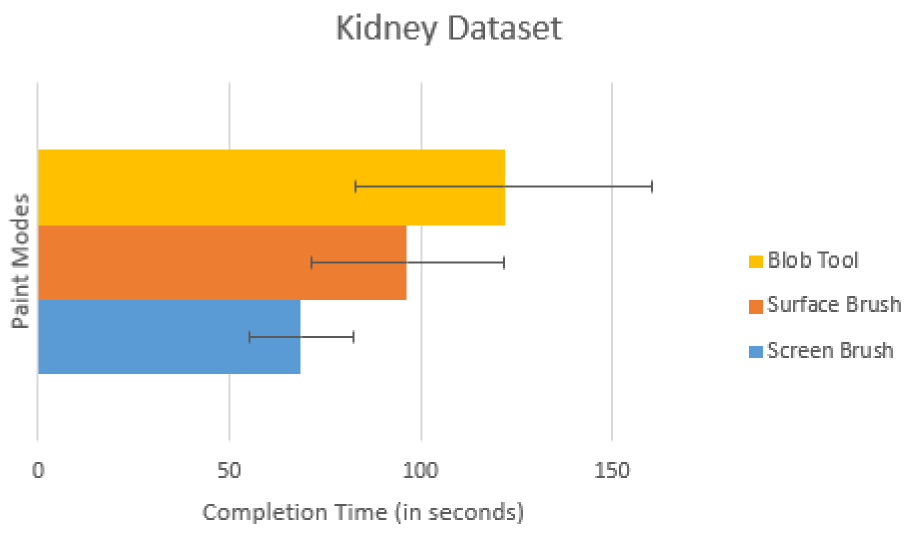


Figure 40: *Average task completion time (in seconds) for each selection technique on the kidney dataset. Error calculated with a 95% confidence interval.*

For the kidney dataset (Figure 40, Table 4.1), many participants struggled with positioning the blob tool in depth using the translation handles. This may indicate that the current implementation of the translation handles wasn’t providing enough of a visual depth cue and/or intuitive positioning control. Participants instead preferred to use the surface brush’s mouse wheel functionality in painting-plane mode for translating the brush in depth based on the camera view. The average completion time reflects this issue as the blob tool took an average of 122 seconds to complete with a 39 second variance calculated at a 95% confidence interval, while the surface brush's average completion time was 96 seconds with a 25 seconds variance calculated at a 95% confidence interval. The average completion time for the screen brush was 69 seconds and participants did better than the two previous techniques due to the kidney's relatively compact area of interest and few surrounding objects. These traits allowed participants to quickly select the kidney while also limiting the amount of spilled paint, thus lowering erasing time significantly.

4.2.1.2 Sternum Results

STERNUM DATASET

TECHNIQUE	Time	CI	F1	CI
BLOB TOOL	48s	[35 - 61]	0.97	[0.96 - 0.98]
SURFACE BRUSH	25s	[21 – 30]	0.79	[0.76 – 0.81]
SCREEN BRUSH	61s	[51 – 72]	0.9	[0.81 – 1.0]

Table 4.2: *Numerical values of the average task completion time (in seconds) for each selection technique on the sternum dataset. Error calculated with a 95% confidence interval.*

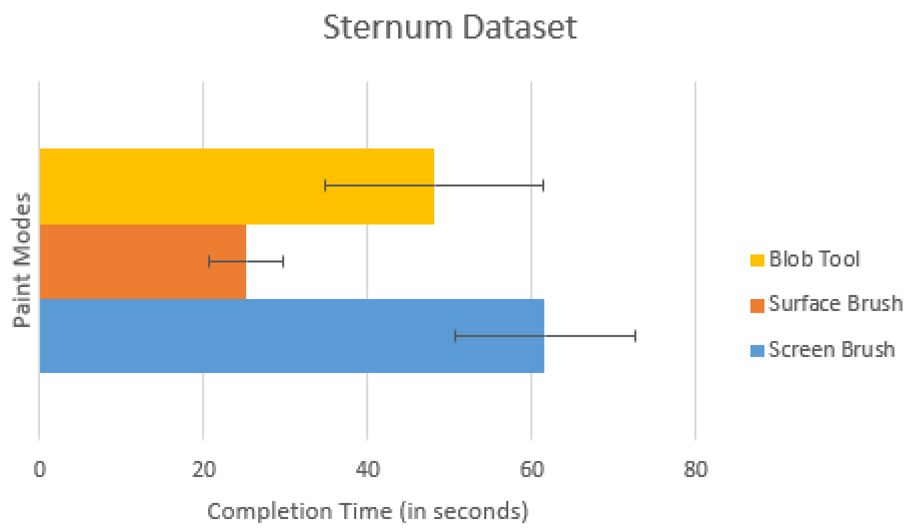


Figure 41: *Average task completion time (in seconds) for each selection technique on the sternum dataset. Error calculated with a 95% confidence interval.*

The sternum dataset (Figure 41, Table 4.2) was the easiest object to select for participants with an average completion time of 25 seconds for the surface brush, 48 seconds for the blob tool, and 61 seconds for the screen brush. The screen brush performed poorly because there are many objects hidden behind the sternum which resulted in participants having to repeatedly rotate the view and perform erasing actions to remove spilled paint. It was predicted that the surface brush would be simple to use for the sternum selection. Using the mouse, the surface brush could easily slide along the sternum's exposed surface and only a few resize operations were needed as the sternum had a fairly consistent thickness. The surface brush was therefore the fastest technique. On the other hand, because of the ease of applying the surface brush to the sternum, it was observed that participants seemed reluctant to rotate the view to check for spilled paint and undo the paint blob or resize the brush before applying it. This reluctance resulted in a rather poor selection accuracy, with an F1 score of 0.79.

Like the surface brush, one option for positioning the blob tool is to simply click on the sternum surface point and the blob tool would automatically jump to that point. However, instead of using a smaller blob size and quickly painting multiple times with this fast positioning capability, many participants opted to use the resize widget handles to make nicely shaped blob to paint the sternum in the fewest paint actions possible. This strategy resulted in slower selection times than the surface brush. However, this strategy is perhaps not unexpected as the idea of the widget interface is the handles suggest their function and operation to the user. Furthermore, while the resizing approach to selection resulted in a higher average completion time, it also produced a significantly higher F1 score with the blob tool F1 score of 0.97 versus surface brush's F1 score of 0.79. Figure 42 illustrates a random participant's selection results for the sternum dataset.

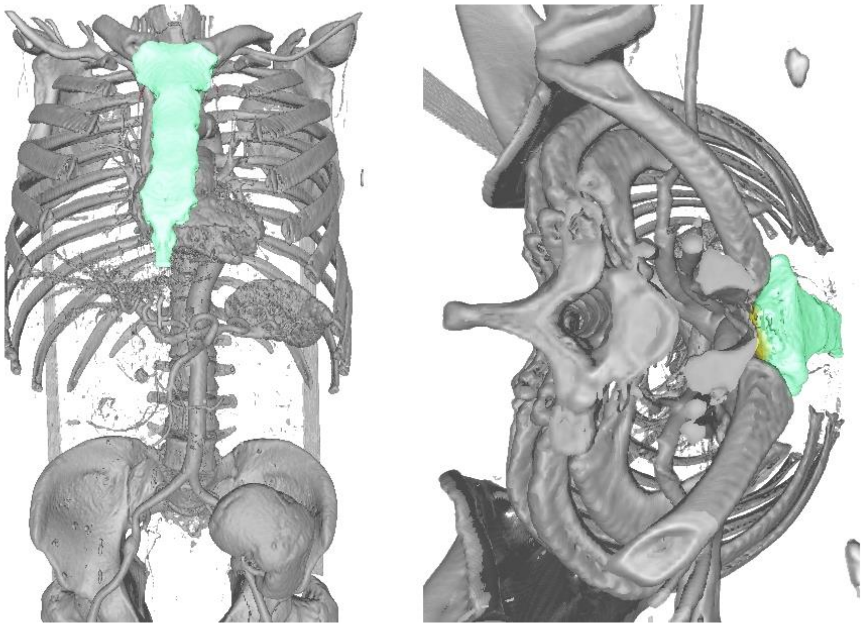


Figure 42: *The front, top, and back view of a randomly selected participant's selection results for the sternum dataset. The green highlight represents correctly selected voxels, the yellow highlight represents target voxels, and the red highlight represents incorrectly selected voxels.*

4.2.1.3 Vertebra Results

VERTEBRA DATASET

TECHNIQUE	Time	CI	F1	CI
BLOB TOOL	70s	[58 - 82]	0.94	[0.9 - 0.97]
SURFACE BRUSH	93s	[69 – 116]	0.82	[0.79 – 0.85]
SCREEN BRUSH	57s	[45 – 70]	0.92	[0.86 – 0.98]

Table 4.3: *Numerical values of the average task completion time (in seconds) for each selection technique on the vertebra dataset. Error calculated with a 95% confidence interval.*

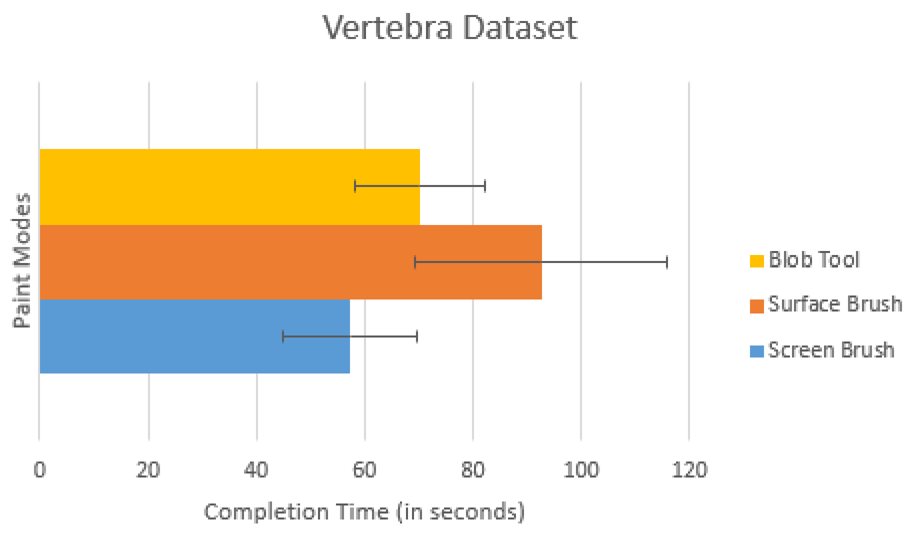


Figure 43: *Average task completion time (in seconds) for each selection technique on the vertebra dataset. Error calculated with a 95% confidence interval.*

The vertebra dataset (Figure 43, Table 4.3) was the most difficult selection task for the surface brush because the dimensions and orientation of the vertebra required the participants to resize and rotate the brush to match it closely. The surface brush used a GUI slider-based brush resize and brush orientation controls on a panel closely adjacent to the volume rendering window. It was observed that participants struggled to connect their GUI slider resize actions to the brush resizing. For example, participants would often lose track of the orientation of the width, height, and depth resize sliders after a rotation slider had been used and this resulted in participants having to test each resize slider again to reorient themselves. This disconnect between the GUI sliders and their effect on the brush seemed to frustrate most of the participants and resulted in a premature termination of the vertebra selection task when using the surface brush. Consequently, the F1 accuracy score was 0.82 with an average completion time of 93 seconds.

The blob tool fared much better as the resize handles provided an excellent visual orientation cue. When a user uses the rotation handles to orient the blob, the resize handles are also reoriented to remove any confusion on how a handle interacts with the blob's current state. The use of the resize handles allowed participants to accurately select the vertebra and resulted in an F1 score of 0.94 with an average completion time of 70 seconds. However, the screen brush had the fastest average completion time of 57 seconds due to the small size of the vertebra and the few surrounding objects. These two traits allowed the user to select the vertebra with minimal spilled paint which resulted in a low editing time and an F1 score of 0.92. Figure 44 illustrates a random participant's selection results for the vertebra dataset.

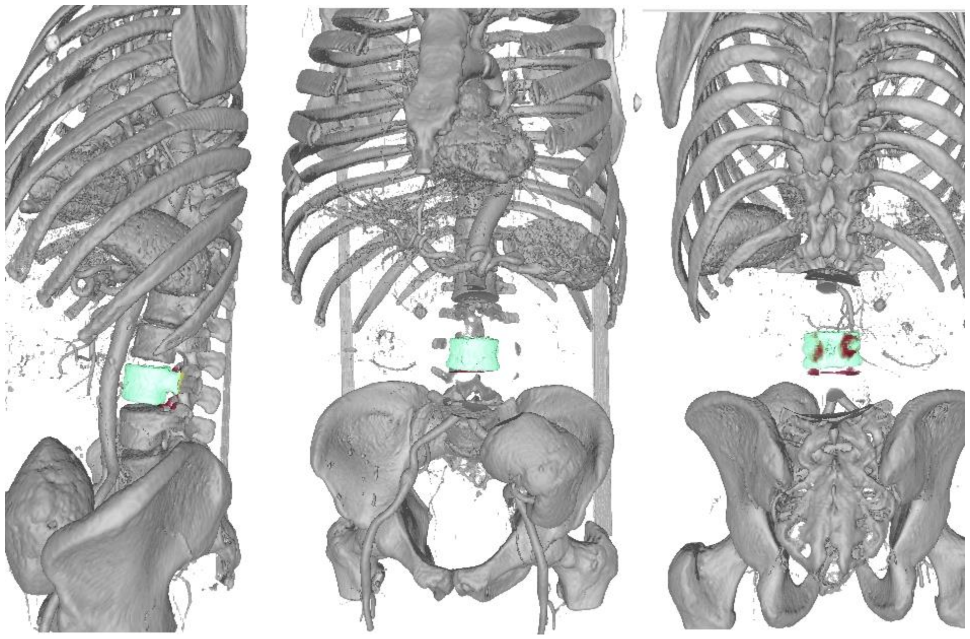


Figure 44: *The side, front, and back view of a randomly selected participant's selection results for the vertebra dataset. The green highlight represents correctly selected voxels, the yellow highlight represents target voxels, and the red highlight represents incorrectly selected voxels.*

4.2.2 Part II: Blob region grow tool

4.2.2.1 Aneurysm Results

ANEURYSM DATASET

TECHNIQUE	Time	CI	F1	CI
STANDARD REGION GROW	59s	[48 – 70]	0.92	[0.9 - 0.94]
BLOB REGION GROW	36s	[29 – 43]	0.95	[0.94 – 0.96]

Table 4.4: *Numerical values of the average task completion time (in seconds) and F1 score for the aneurysm dataset. Error calculated with a 95% confidence interval.*

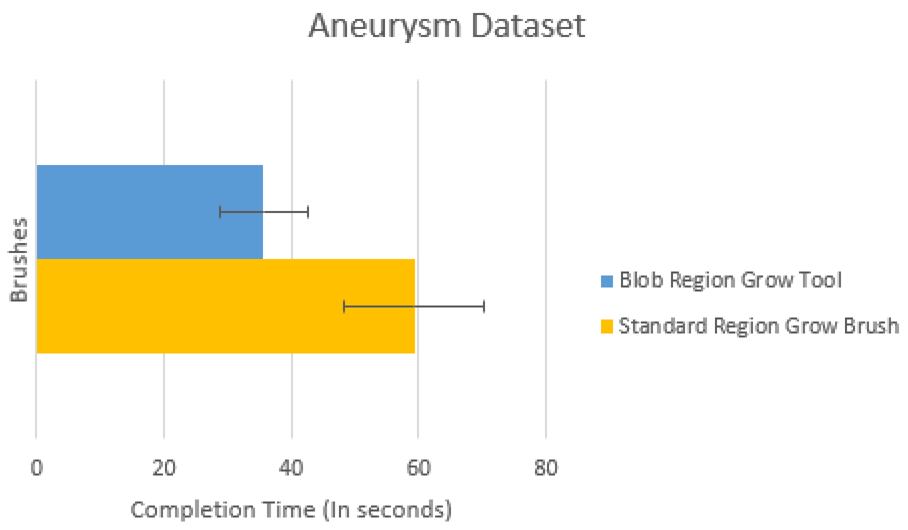


Figure 45: *Average task completion time (in seconds) and F1 score for the aneurysm dataset. Error calculated with a 95% confidence interval.*

The aneurysm dataset (Figure 45, Table 4.4) was selected to evaluate the usefulness of our 3D widget-based UI for the blob region grow tool. The goal was to determine if the widget handle UI improved selection performance and user satisfaction when compared to a region grow technique with a more standard and simpler mouse-based UI. With the standard mouse-based UI, users began by selecting an initial seed voxel. Then, by simply moving the mouse away or back towards this seed voxel position, the selected region expands or contracts uniformly in all directions. That is, the selected region has a spherical shape only – unlike the super-ellipsoid blob region grow tool. This simple but often effective UI design was patterned off the region grow tool of the publicly available LiveVolume [8] volume rendering software system.

In order to test the hypothesis that a widget based, super-ellipsoid constrained UI would have superior performance, we chose a target object that could, in theory, be easily selected by the standard region grow tool by naïve users. That is, rather than choosing a more complex-shaped target object that might bias the experiment toward the more flexible UI of the widget-based blob region grow tool. The aneurysm (Figure 46) has a relatively spherical shape that can be selected in a single grow interaction if the initial seed voxel is carefully chosen. However, all participants were unable to do so, due to poor initial seed placement. Without the extra controls provided by the widget handles, accurate selection depended on the initial seed placement and editing the selection with a uniform grow in all directions resulted in "paint" spilling onto connected non-target structures. Conversely, with the widget-based blob region grow tool, participants were able to place the initial seed point anywhere in the dataset and quickly adjust the selected region boundaries independently on six sides using the widget handles. As a result, users performed much better with the widget-based blob region growing tool with an average completion time of 36 seconds and an average F1 score of 0.95. In comparison, the standard region grower's average completion time was 59 seconds and average F1 score of 0.92. In a questionnaire asking participants to name their favorite UI between the two region grow tools, the blob region grow tool was unanimously the favorite. Participants commented that the widget handles were intuitive and added flexibility.

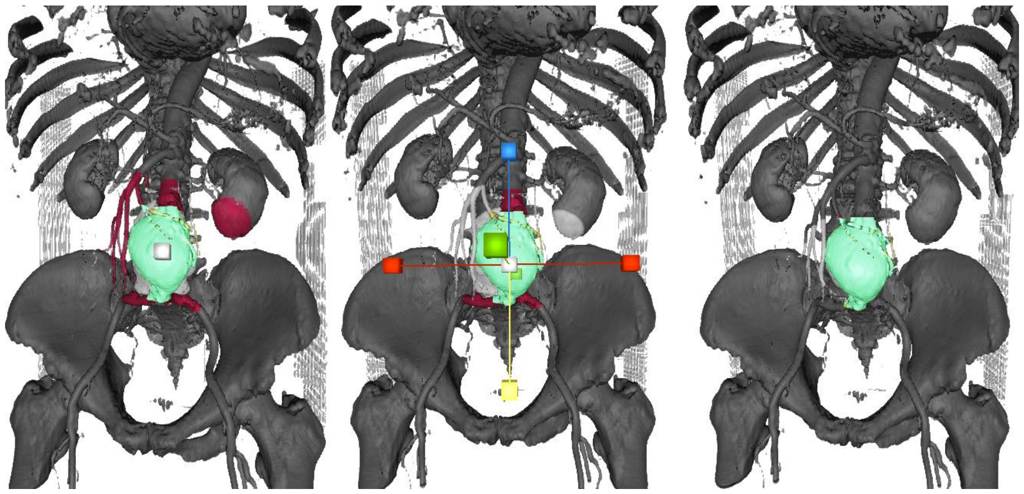


Figure 46: *A comparison of a standard region grow tool and blob region grow tool. Left: Standard region grow control that grows/shrinks uniformly in all directions based on mouse movement. Middle: blob region grow that can be resized independently using the widget handles. Right: A participant's selection results for the aneurysm dataset using the blob region grow tool. The green highlight represents correctly selected voxels, the yellow highlight represents target voxels, and the red highlight represents incorrectly selected voxels.*

4.2.2.2 Double Vertebra

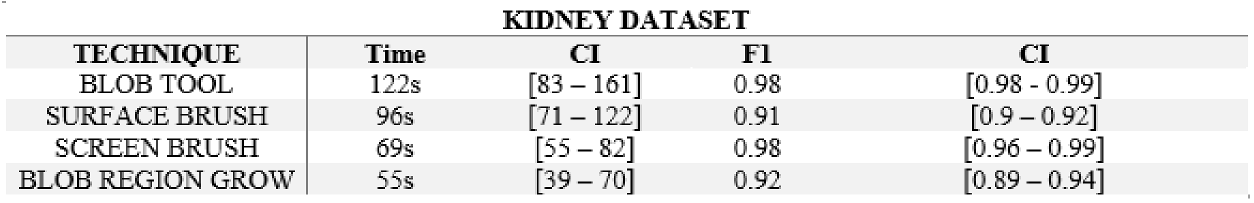
DOUBLE VERTEBRA

TECHNIQUE	Time	CI	F1	CI
BLOB REGION GROW	93s	[82 – 104]	0.88	[0.78 - 0.98]

Table 4.5: *Numerical values of the average completion time (in seconds) and F1 score for the blob region grow tool on the double vertebra dataset. Error calculated with a 95% confidence interval.*

The double vertebra dataset (Table 4.5) was only tested with the blob region grow tool due to time constraints. However, further insight into the effectiveness of the widget interface was gleaned. All blob region grow experiments were conducted last and therefore resulted in a lower learning curve for participants due to their experience in the previous trials with the widget interface. To counter this advantage, participants were asked to select two vertebrae with the blob region grow tool. However, although most participants accurately selected the surface of each vertebra using the blob region grow tool, unfortunately voxels inside the vertebra were missed resulting in an F1 accuracy score of 0.88. This is a potential problem with region growing, as mentioned previously, as it selects voxels within a specified range that are connected to the initial seed voxel. Many objects have a large intensity variation in their interior, resulting in disconnected islands of valid voxels. Nonetheless, it is interesting to note that both vertebras were selected with an average completion time of 93 seconds. Therefore, a single vertebra was selected in roughly 47 seconds (50% of 93) – which is considerably lower than the fastest technique in the single vertebra experiment (57 seconds using the screen brush).

4.2.2.3 Blob region grow tool vs Blob tool, Surface brush, and Screen Brush



KIDNEY DATASET

TECHNIQUE	Time	CI	F1	CI
BLOB TOOL	122s	[83 – 161]	0.98	[0.98 - 0.99]
SURFACE BRUSH	96s	[71 – 122]	0.91	[0.9 – 0.92]
SCREEN BRUSH	69s	[55 – 82]	0.98	[0.96 – 0.99]
BLOB REGION GROW	55s	[39 – 70]	0.92	[0.89 – 0.94]

Table 4.6: *Numerical values of the average task completion time (in seconds) for each selection technique on the kidney dataset with blob region grow tool included. Error calculated with a 95% confidence interval.*

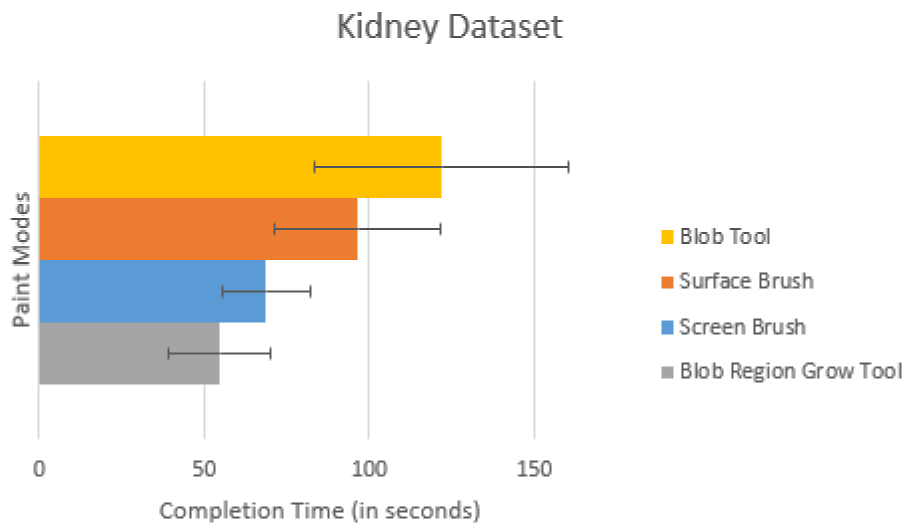


Figure 47: *Average task completion time (in seconds) for each selection technique on the kidney dataset with blob region grow tool included. Error calculated with a 95% confidence interval.*

One experiment was performed to compare the blob region grow tool with the purely geometric region selection tools using a single anatomical structure – the kidney transplant. While the blob region grow tool is somewhat different in nature than the geometric tools in that it selects *connected* sets of voxels starting from a seed voxel, it was nonetheless deemed useful to gain further insight into the widget-based UI and into the region-grow tool (Figure 47, Table 4.6). As mentioned previously, to create a user study for naïve users, the target ROIs used in the study were all single anatomical structures (i.e. consisting of connected voxels). Therefore, it was expected that the blob region grow tool would outperform the other tools for these structures, at least in terms of selection time. Indeed, the blob region grow tool resulted in the fastest selection time with an average completion time of 55 seconds and a variance of 16 seconds calculated at a 95% confidence interval. Participants primarily used the resize widget handles to quickly grow the selected voxels into the entire kidney. As the region growing algorithm doesn't grow on already selected voxels, participants also would often perform smaller region grow selections and use them as boundaries to perform a large selection on the entire kidney without “spilling” paint into surrounding areas. When selecting connected sets of voxels, the blob region grow tool has an inherent advantage as it lowers the importance of having to position the blob accurately to select the target region. Most geometric-based techniques require more precise positioning. Furthermore, often participants opted to ignore the translation handles of the blob region grow tool completely and exclusively use the resize widget handles. This interaction strategy dramatically reduced the average selection completion time and may suggest that the resize handles are more effective in general for selection tasks as the handles (and hence the blob resizing) can be controlled independently on each side and therefore can also act secondarily to translate the blob. Unfortunately, a weakness of the blob region grow tool is that it may miss voxels underneath the surface and resulted in a F1 score of 0.92. However, if voxels beneath the surface is not important, the blob region grow tool is just as effective as the blob tool and screen brush at capturing the shape of an object. Interestingly, the accuracy of the surface brush also suffered the same issues as the blob region grow tool due to user’s reluctance to resizing the brush to encapsulate entire regions (F1 score of 0.91). Users instead opted to paint multiple blobs along the surface of the kidney with a relatively small brush which resulted in voxels underneath the surface of the kidney to be missed. The accuracy of the blob tool (F1 score of 0.98) didn’t suffer this issue due to user’s preference of using the blob tool’s widget resize handles to encapsulate entire sections of the kidney before painting.

4.2.3 Questionnaire Results

After the trials for each tool were completed, participants were asked to fill out a questionnaire to evaluate their level of satisfaction for each technique. Each question was rated on a 7-point

Likert scale with a 1 indicating the lowest level of satisfaction and a 7 representing the highest. Users were also asked to pick their favorite technique before and after the introduction of the blob region grow tool (Figure 48,49). Before the introduction of the blob region grow tool participants generally favored the screen brush as it provided a simple and familiar interface for the selection tasks. The blob tool came in second as users with gaming or 3D modeling experience were familiar with the handle-based controls. The surface brush was the favorite for users that preferred to only use the more direct mouse-dominant surface sliding interface. It was observed that these users generally struggled with spatial awareness as well as coordinating between the mouse and keyboard, which put the other techniques at a disadvantage. Participants overwhelmingly favored the blob region grow tool once it was introduced. The intuitive blob tool interface combined with the region-grow functionality allowed users to quickly and accurately select a single object ROI. The least favorite technique was the surface brush (Figure 50), most likely due to having the brush size and shape controls on a separate GUI panel, which frustrated users in tasks where the brush required resizing and/or reorienting. This result was expected for these tasks as the surface brush, as mentioned, is a more direct manipulation technique for *positioning* and automatic orientation. However, the cost of this direct-manipulation capability is the difficulty of integrating resizing and orientation fine tuning into the interface. Several special keys can be used and combined with mouse movement or the mouse scroll wheel, but the resulting UI is rather clumsy and requires memorizing the keys. On the other hand, when using the surface brush painting-plane mode, the functionality of using the mouse wheel to move the brush in depth, dependant on the user’s view, was well received and should be incorporated into the blob tool in future work. Each technique was also evaluated on how easy it was to learn (Figure 51), how easy it was to control (Figure 52), and how easy it was to select objects (Figure 53). Participants found the screen brush easiest to learn with an average score of 6.29 followed by the blob region grow tool at 5.66, the surface brush at 5.31, and the blob tool at 5.06. For control, participants preferred the screen brush with an average score of 6.47 followed by the blob region grow tool at 6, the surface brush at 5.12, and the blob tool at 4.87. For selection, the screen brush came out on top again with an average score of 6 followed by the blob region grow tool at 5.91, blob tool at 5.12, and surface brush at 4.5. Interestingly, participants rated the blob tool relatively poorly when compared to the widget blob region grow tool even though they both shared the same interface. This difference in rating may have resulted from users performing the blob region grow tool trials last and thus giving them an experience advantage. Generally, the open-view functionality was well received and was rated on how helpful the functionality was for viewing and selecting hidden objects with an average score of 5.5 and 5.37 respectively (Figure 54). Lastly, our blob region grow tool was unanimously the favorite when compared against a standard uniform region grower. Participants commented that the widget handles were intuitive and added flexibility. These traits resulted in participants selecting the aneurysm with an average completion time of 35.6 seconds and an average F1 score of 0.95. Versus the uniform region grower’s average completion time of 59.25 seconds and average F1 score of 0.92 (Figure 55).

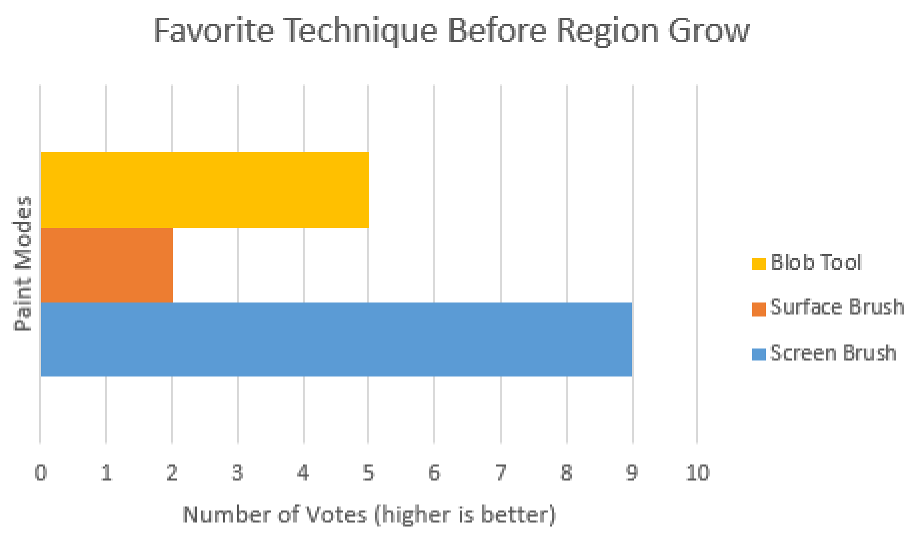


Figure 48: *Participant's favorite technique excluding blob region grow tool.*

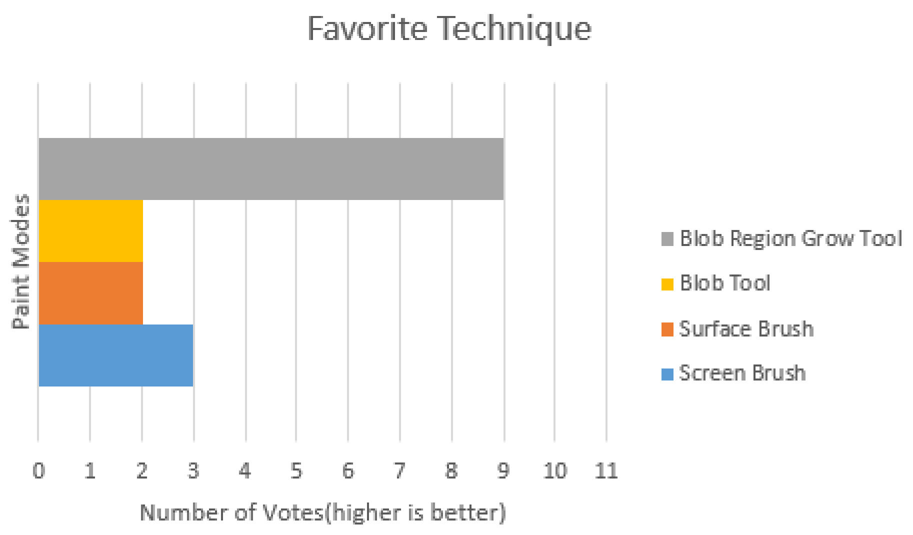


Figure 49: *Participant's overall favorite selection technique.*

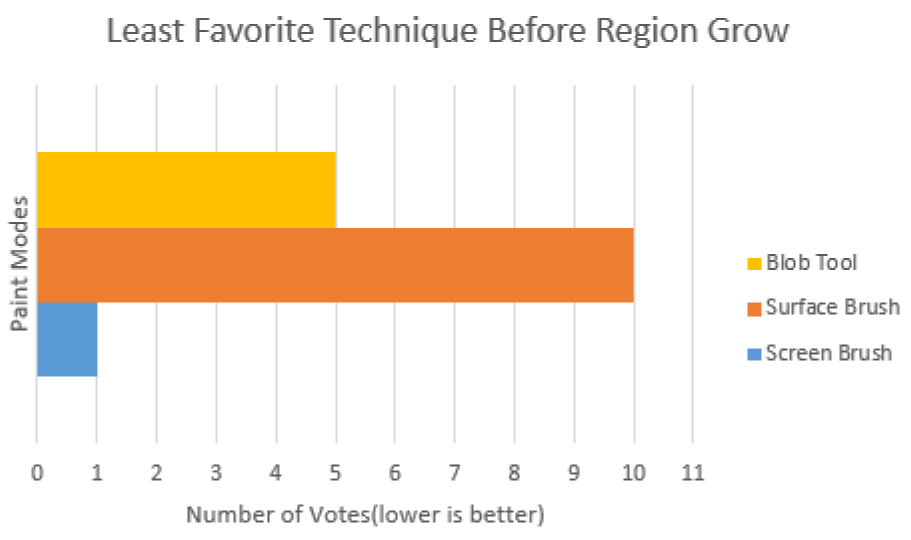


***Figure 50:** Participant's overall least favorite technique.*

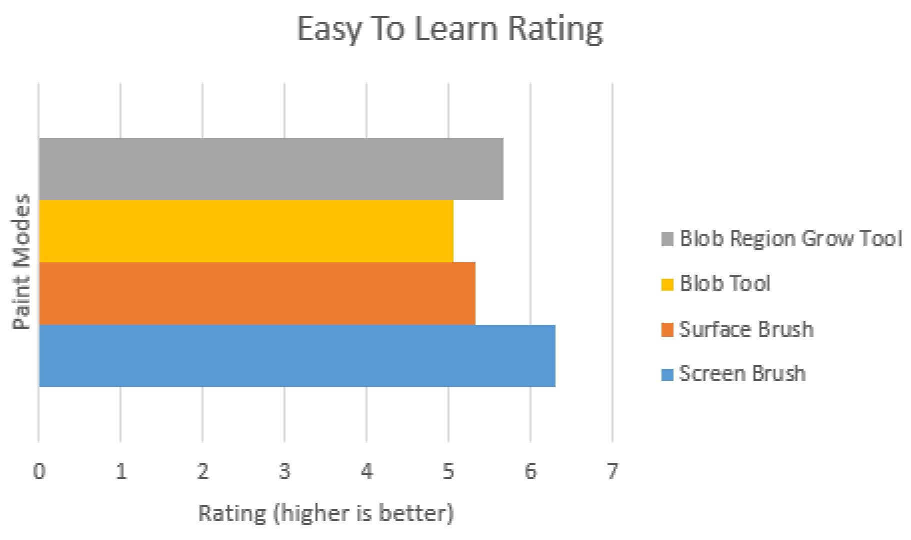


***Figure 51:** Participant's rating on how easy each technique was to learn.*

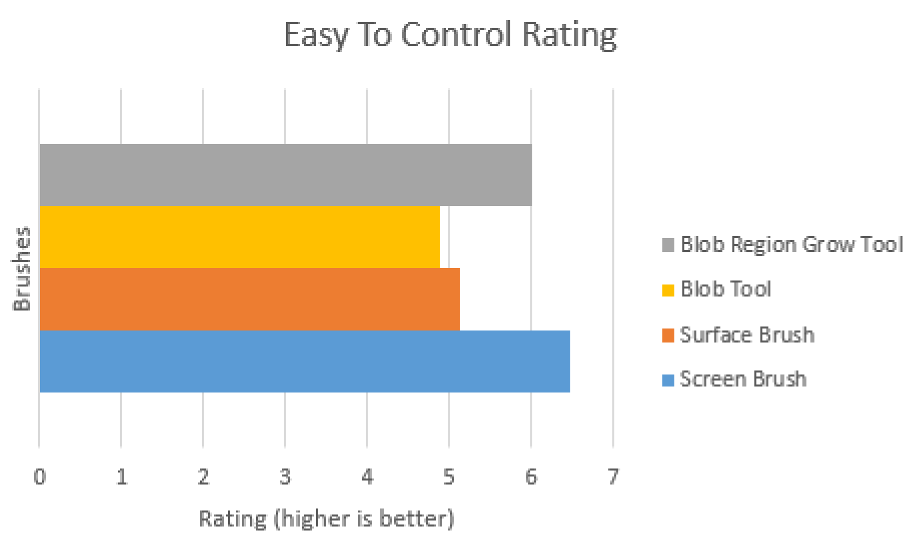


Figure 52: *Participant's rating on how easy it was to manipulate the tools into their desired form via translating, resizing, or rotating.*

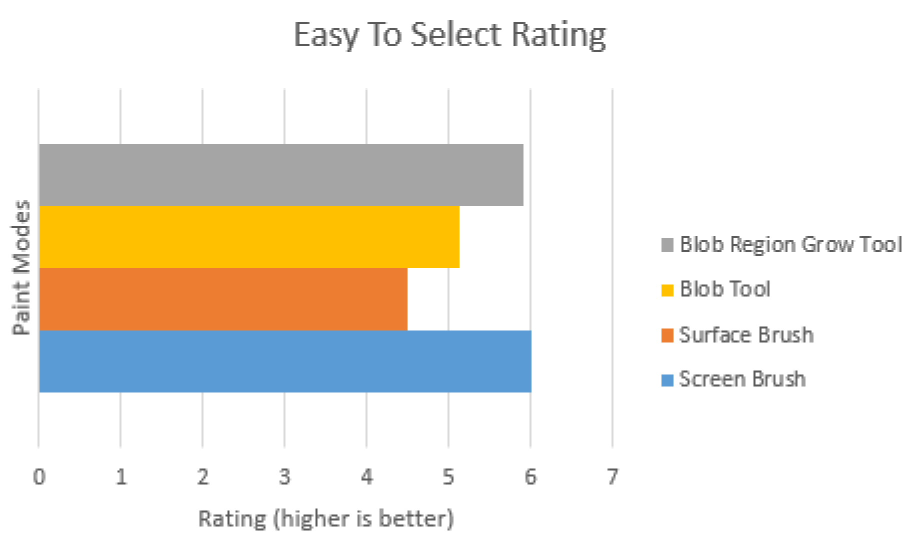


Figure 53: *Participant's rating on how easy it was to manipulate the tools and select their desired region of interest.*

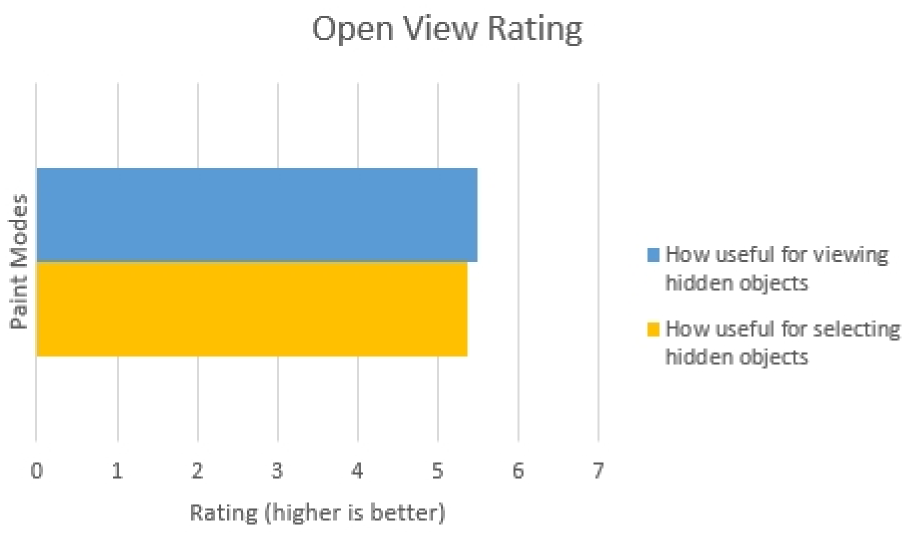


Figure 54: *Participants rating on how useful open view was for selecting and viewing hidden objects.*

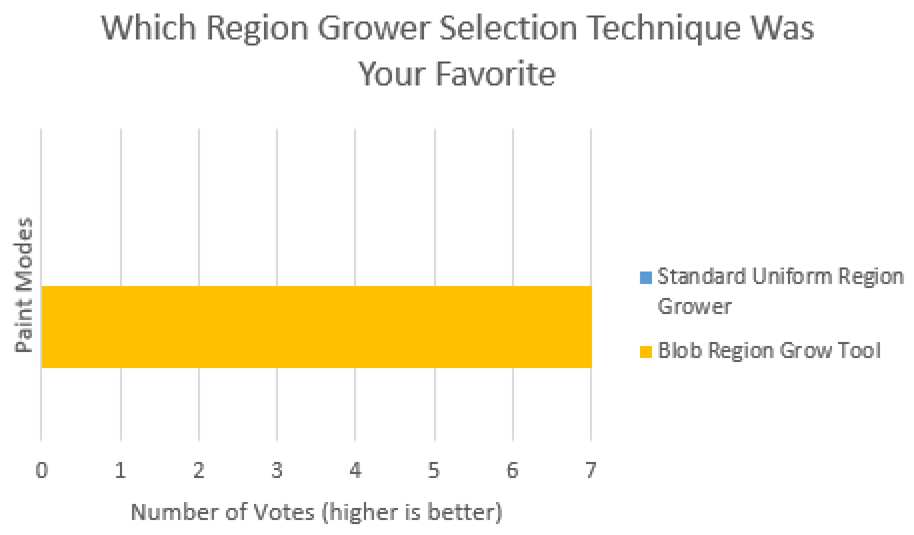


Figure 55: *Participant's favored region growing selection technique.*

4.3 Additional Experiments

Additional experiments were performed to demonstrate the capabilities and flexibility of the system in various volume image selection/exploration situations. All experiments were performed in less than 5 minutes and a brief description and discussion of the selection process is provided for each experiment.

4.3.1 Dilated Aorta Experiment

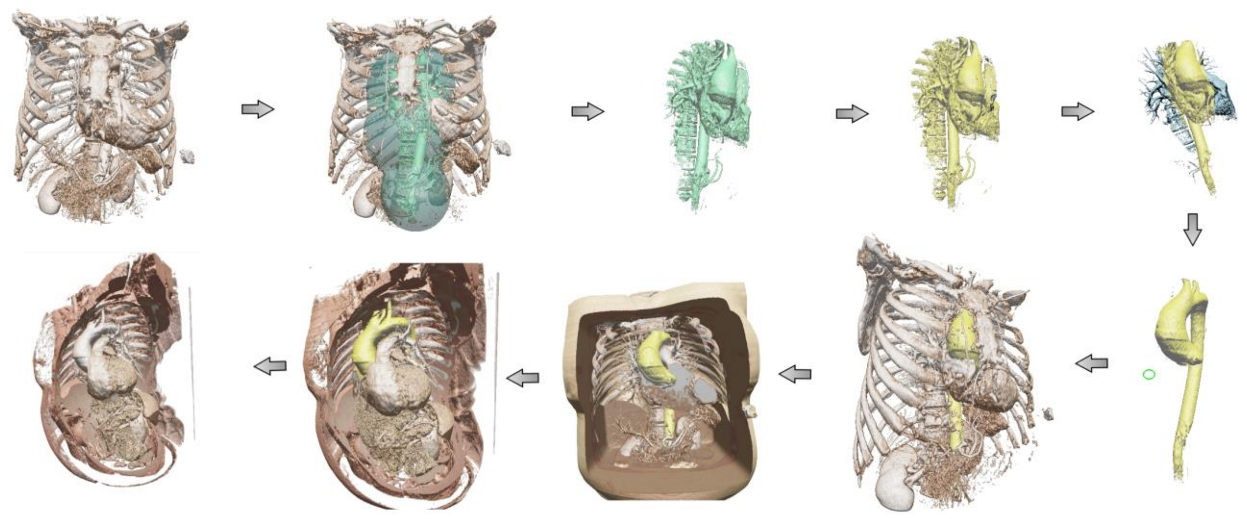


Figure 56: *CT data set for a patient with a dilated aorta. Starting from the top left: 1. Set initial minimum and maximum visibility settings to view aorta and surrounding structures without losing any detail. 2. Create a ROI using the blob tool. 3. Add voxels selected in the ROI to the accumulation grid. 4. Erase voxels from accumulation grid with screen brush. 5. Restore original view. 6. Add open view to create various contextual views by changing the viewing angle and visibility settings.*

Figure 56 is a series of volume renderings of a CT data set (down-sampled to 256x256x170 voxels) for a patient with a dilated aorta. The goal was to select the aorta and create views to allow it to be viewed in context and measured. In this experiment, a combination of the blob tool and screen brush is used. The top left of Figure 56 shows a view of the dataset with a voxel visibility range that removes as much of the surrounding information as possible, while also maintaining the details of the aorta. A simple 1D TF was used to create this view. To remove the remaining occluding objects, the blob tool was used to plant a series of blended blobs to roughly isolate the region surrounding the aorta. A separate TF controlling the visibility of voxels outside the ROI was then configured to hide all voxels. The selected voxels within the ROI were then added to accumulation grid (Figure 56, top row, second from right) and unwanted voxels were removed using the screen brush’s eraser function (Figure 56, top and bottom rows, right). Once the segmentation of the aorta was complete, the voxel visibility range was adjusted to re-display the surrounding context. An open-view lens was then added to create several contextual views, each using a different lens shape. The open-view lens generates a cross-sectional view within the heart region that allows users to more clearly view the aorta with respect to the heart. The open-view lens can be reoriented and reshaped in real-time and if desired, the aorta itself can be cut in two and its cross-section measured.

4.3.2 Aneurysm Experiment

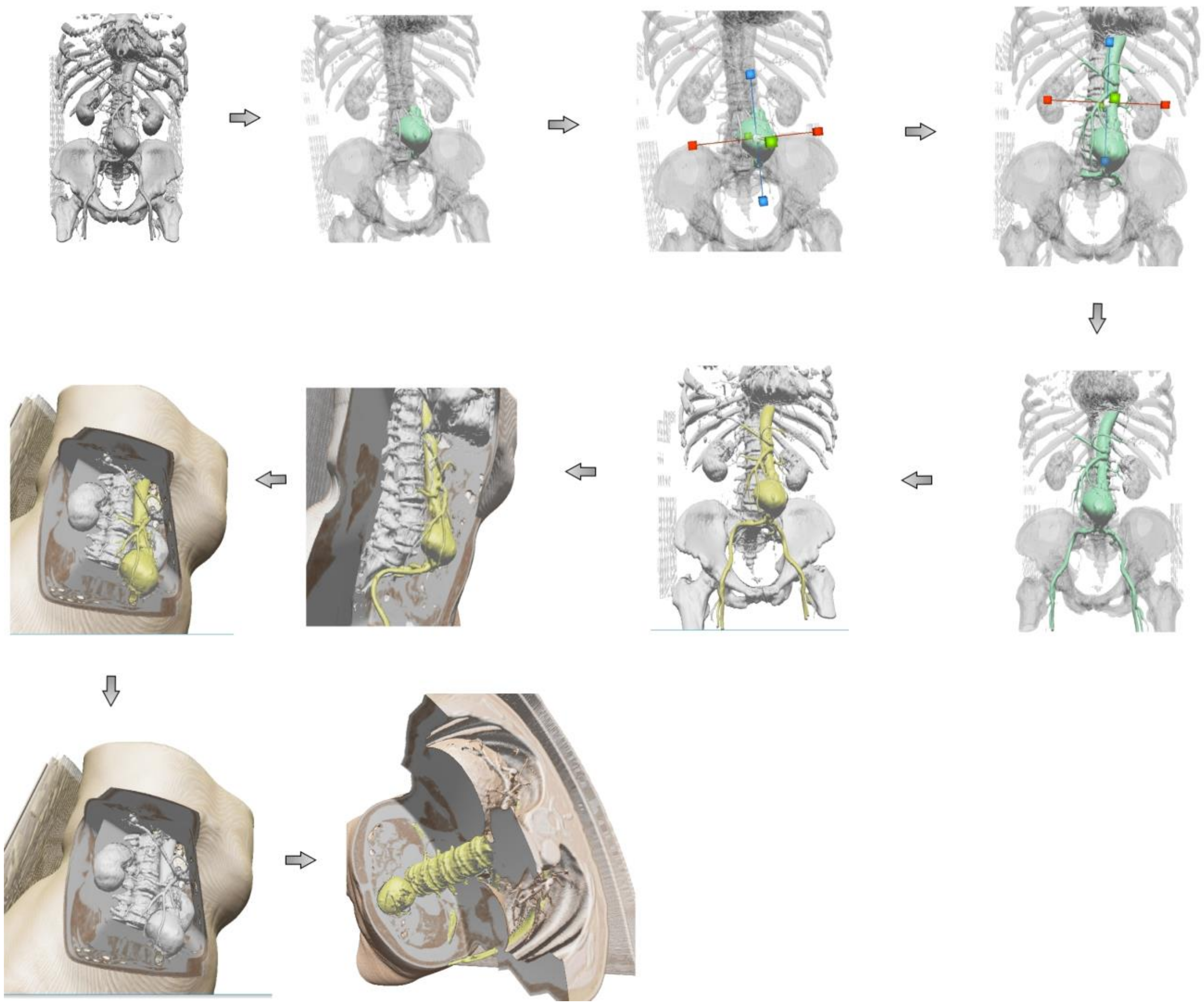


Figure 57: *Aneurysm selection experiment. Starting from the top left: 1. Set minimum and maximum visibility to view aneurysm and surrounding structures. 2. Select aneurysm with blob region growing tool and interactively adjust the blob to add connecting arteries. 3. Add selected voxels to accumulation grid. 4. Add open-view lens to create various contextual views by changing the viewing angle and visibility settings.*

The region of interest for this CT dataset (down-sampled to 256x256x144) is the aneurysm and the connected arteries (Figure 57). An aneurysm is an excessive localized enlargement of an artery cause by a weakening of the artery wall. Like the previous experiment, an initial voxel visibility intensity range was chosen that removes as much of the surrounding information as possible without removing any important detail from the ROI. As the data set was relatively clean (i.e. noise free), the blob region grow tool was used to initially select the aneurysm itself. At this point, the aneurysm volume can be measured, if desired. The widget handles were then used to resize the region grow blob to grow onto connecting arteries and the selected voxels were added to the accumulation grid. The minimum voxel visibility range was then lowered to render surrounding skin, muscles, and bones. A rectangular open-view lens was then added to create a view that allows users to clearly see the aneurysm's position in relation to various surrounding structures. The open-view lens also allows the user to view the aneurysm from any angle and quickly gain additional insight.

4.3.3 MRI Brain Tumor

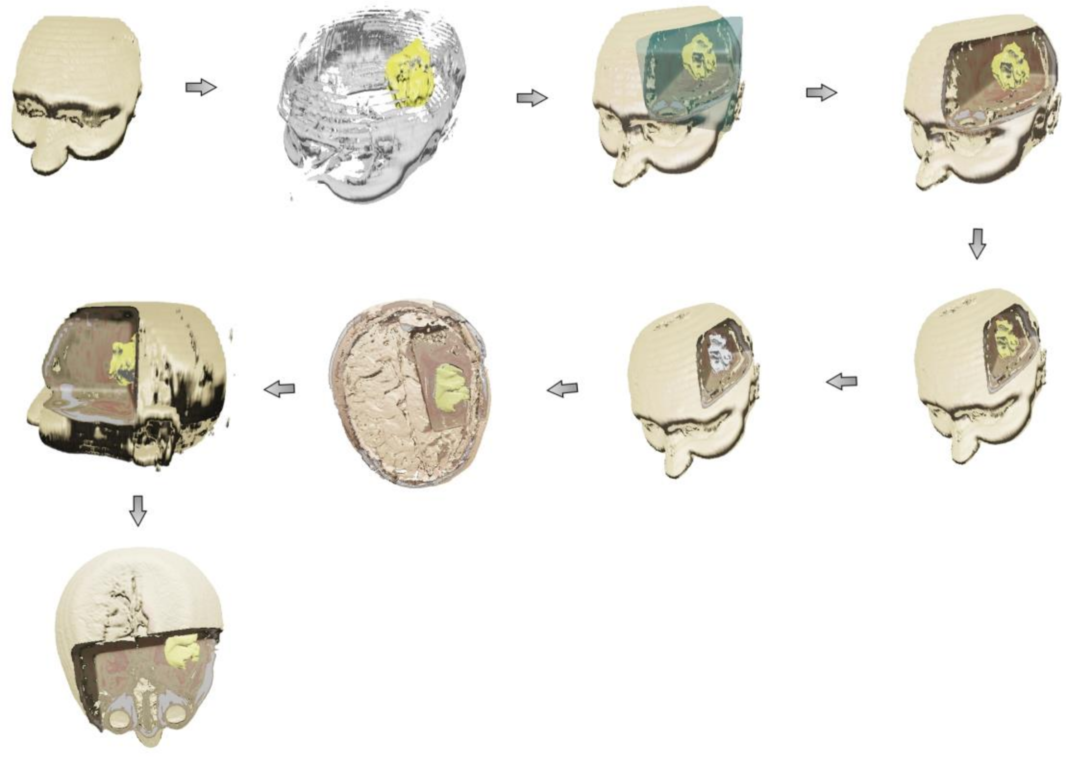


Figure 58: *MRI brain tumor selection experiment. Starting from the top left: 1. Set initial minimum and maximum visibility settings to view brain tumor without losing any detail. 2. Select tumor with blob region grow tool and add to accumulation grid. 3. Use blob tool and open view to create various contextual views by changing the viewing angle and visibility settings.*

This experiment uses a (down-sampled) 288x288x22 MRI scan and demonstrates how the system can be used with nosier volume images to create effective views (Figure 58). In this case, the object of interest is a tumor inside the brain with a very high intensity value. To select the tumor, the minimum voxel visibility range was set to a high value and the blob region grow tool was used to select and add the tumor to the accumulation grid. To create cross sections in the brain, a rectangular blob tool was used with its minimum voxel visibility range was set to the maximum value. Since voxels within the accumulation grid are not affected by the visibility settings of the blob tool, it allows users to quickly create cross sections by translating, rotating, or resizing the blob. This allows users to quickly ascertain the tumor's position relative to other objects within the dataset.

4.3.4 Hypernephroma Experiment

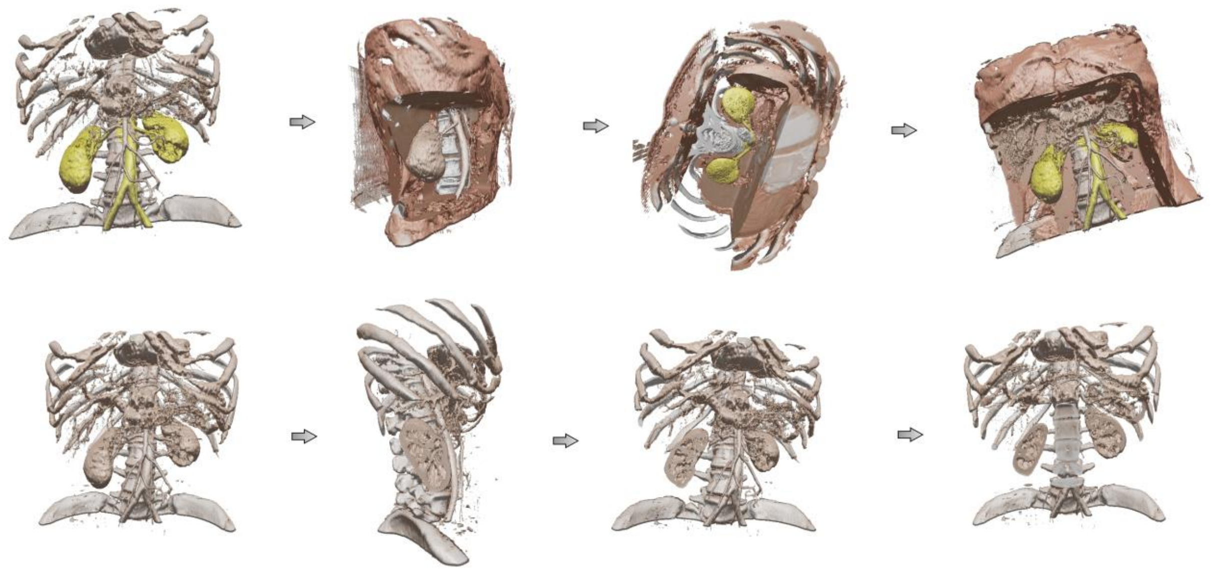


Figure 59: *Hypernephroma exploration experiment. Starting from the top left: 1. Select kidneys and connected arteries using the blob region grow tool and add selected voxels to accumulation grid. 2. Adjust visibility settings via TF to view muscle and organs. 3. Add open-view lens to create various contextual views by changing the viewing angle and visibility settings. The bottom row starting from the left illustrates how open-view can be used to make cut surfaces: 1. Adjust visibility settings to show kidneys. 2. Place a cubical open-view within kidney to create a cut surface. 3. Adjust open-view size and placement to inspect kidney cross-section.*

Hypernephroma is a common type of kidney cancer that begins in the lining of the renal tubules of the kidney. This experiment is a 256x256x117 (down-sampled) CT scan and shows how our system can be used to select the kidneys and its surrounding arteries, or as a tool to create cross sections using an invisible rectangular blob to cut the kidney in half and view the cancer within.

Starting from the top left of Figure 59, the kidneys and its surrounding arteries were added to the accumulation grid using the blob region grow tool. The minimum voxel visibility range was then lowered to show muscles and organs. A rectangular open-view was also added to allow the user to inspect the kidneys from various angles. The bottom row of Figure 59, starting from the left, shows how a rectangular open-view can be used to create cross sections within the kidney to view the pockets within and inspect the areas affected by the cancer.

4.3.5 Pulmonary Stent Experiment

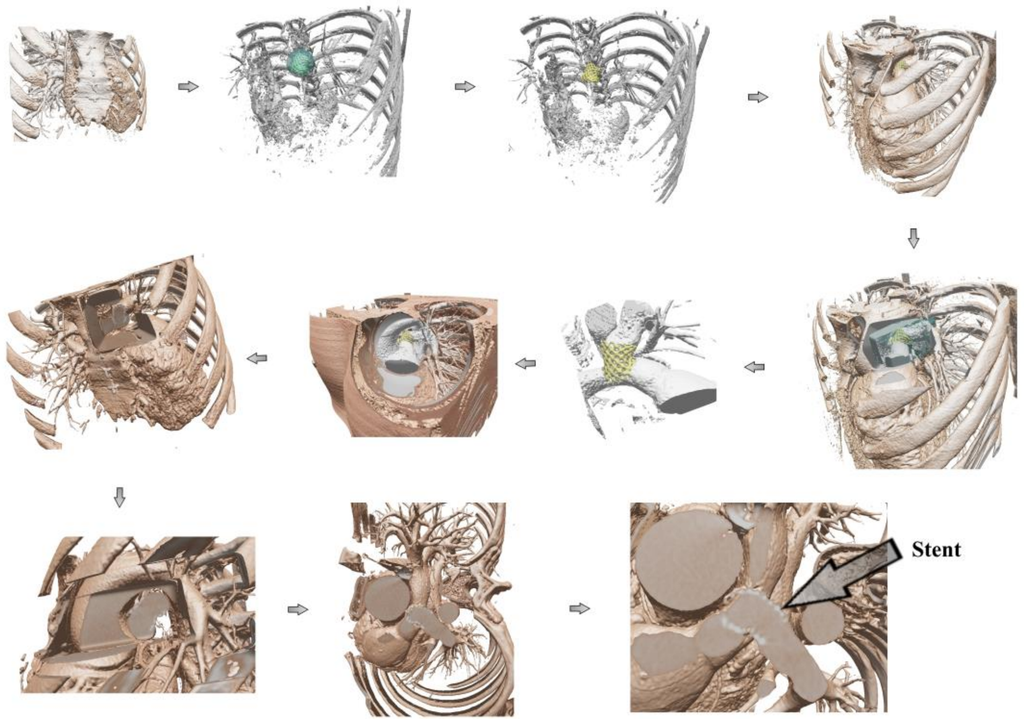


Figure 60: *Pulmonary stent experiment. From top left: 1. Create initial view of stent and lung region using TF. 2. Increase minimum visible voxel intensity via TF to isolate the stent. 3. Use blob tool and paint the stent region and add it to accumulation grid. 4. Use blob tool to create an ROI around the stent and remove surrounding structures. 5. Add open-view lens to create various views by changing the viewing angle and visibility settings.*

A stent is a metal or plastic tube inserted into an airway or artery (the pulmonary artery in this example) to keep it open. This experiment was conducted on a 512x512x308 CT scan and demonstrates how our system can be used to select a stent and inspect its position within the artery. Starting from the top left of Figure 60, the voxel visibility range was adjusted to view the lungs, but unfortunately, the stent wasn't clearly visible until the voxel intensity range was set to a high value. As the surrounding area was clean, the blob tool was used to select the stent and the selected voxels were added to the accumulation grid. The voxel visibility range was then lowered to view the stent's position around the heart and top of the lungs. To get a clearer view, the blob tool was placed around the area of the stent and all outside voxels were removed. This allowed the user to zoom in and create an enlarged view of the stent and its surrounding objects for inspection. Voxels of lower intensity were then brought back, and an open view lens was added to show cross-sectional views of the stent within the artery. These two techniques allow the user to inspect the stent's position relative to other objects and to potentially verify the stent was correctly installed.

4.3.6 Kidney Transplant Experiment

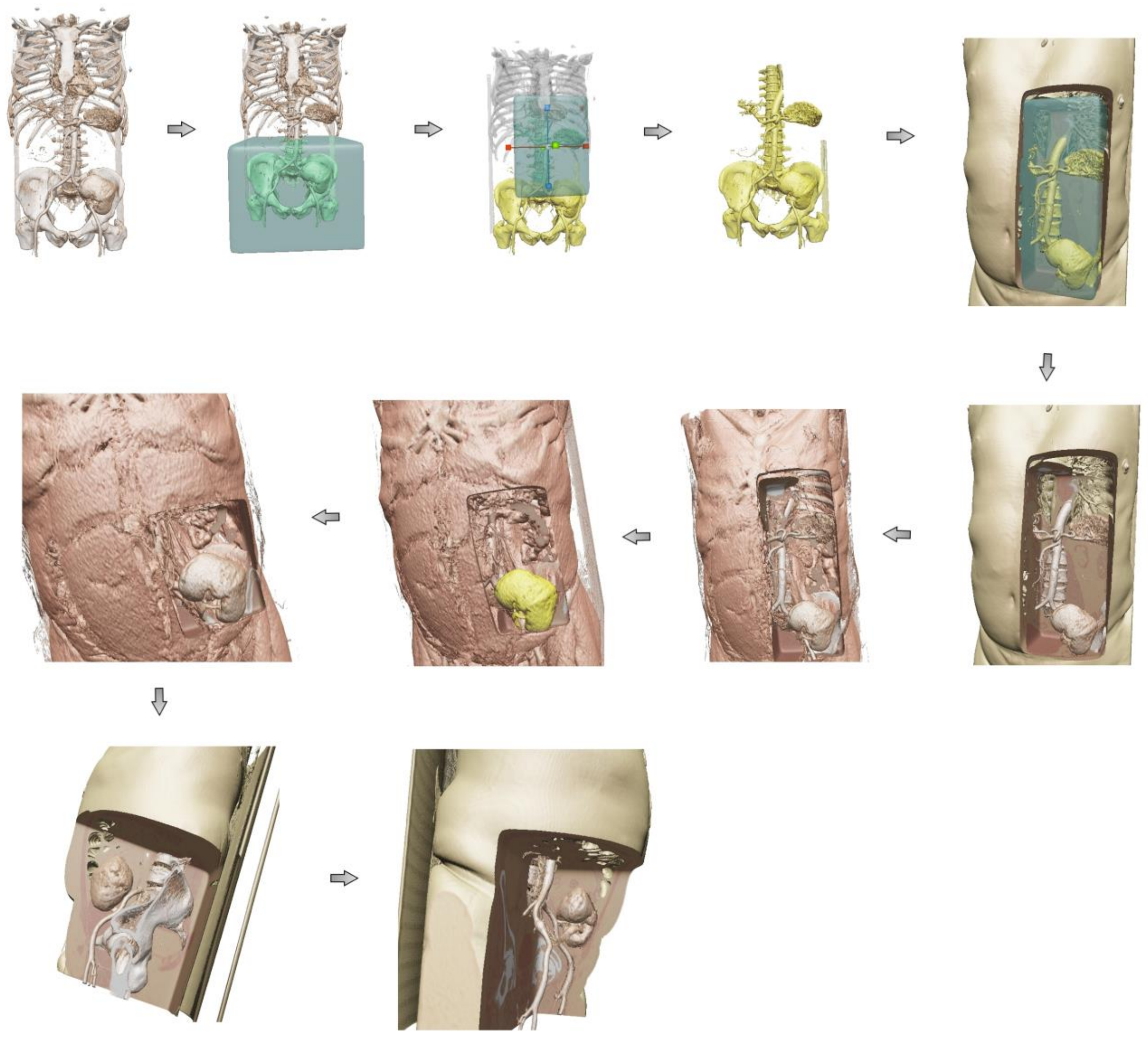


Figure 61: *Kidney transplant experiment. Starting from the top left: 1. Create initial view of bones, arteries, and kidney transplant via TF. 2. Use blob tool to select the hipbone and kidney transplant. 3. Use blob region grow tool to select the connected arteries and the kidney. 4. Use a cubical shaped blob to create a cut away using the blob tool. 4. Create contextual views by changing the viewing angle, blob size, and visibility settings to view the focus region in respect to muscles or skin.*

This experiment was conducted on a 256x256x296 (down-sampled) CT scan and showcases how a user could create various contextual views for a kidney transplant. Starting from the top left of Figure 61, the hip bone and its surrounding structures were selected using the blob tool and the selected voxels were added to the accumulation grid. The remaining connecting arteries and vertebrae were then selected using the blob region grow tool and were also added to the accumulation grid. A rectangular blob tool was then placed over the accumulated paint and the minimum voxel visibility range was set to reveal skin. The minimum voxel visibility range for the blob was then set to the maximum value to hide all voxels within the blob that were not within the accumulation grid. An open-view lens was then added and combined with various blob sizes and visibility settings to create several contextual views.

Chapter 5: Conclusions and Future Work

In this thesis, a volume rendering system that uses super-ellipsoids to perform fast volume of interest selection in medical images via a widget-based interface was presented. It accomplishes this by providing users with a blob tool that can be intuitively translated, scaled, rotated, and transformed. 3D paint, in the form of blended super-ellipsoids, can be deposited to form a well-defined 3D ROI in which a separate transfer function can be applied. The ability to define an area of interest with a separate transfer function using multiple complementary tools allows users to quickly create effective contextual views by breaking the view generation problem into a series of simple and intuitive geometrically-defined interactions. A real-time blob region grow tool with a widget-based UI was described. A screen brush with a simple UI was also integrated into our system as another complementary technique to select or erase regions. An open-view auxiliary lens functionality was introduced as a method for quickly exploring and viewing occluded objects. It accomplishes this by giving users the ability to create view-dependent cutaway regions with a user-adjustable super-ellipsoid defined cap that is attached to the blob tool. This functionality allows users to quickly customize the cutaway region shape to either explore or view occluded objects in any situation. An accumulation grid was introduced to combine the output of the intermediate selections resulting from the individual interactions. Examples of this workflow were presented in a series of experiments in Chapter 4.

The effectiveness of the widget-based UI was quantitatively and qualitatively evaluated in a user study. Users overwhelmingly preferred the widget-based blob region grow tool. The widget-controlled blob tool performed satisfactorily and was the second most preferred geometric selection technique after the screen brush. Observations and feedback from the user study showed that participants had trouble positioning the blob tool in depth and identifying the blob's axis of rotation. This implied that the widget handles for translation and rotation weren't providing enough control and visual depth cue and therefore may have negatively impacted results. Improvement to the widget handle design is planned in future work. The advantage of widgets for manipulating the blob is that they are data independent and can be applied to any volume image and any selection scenario. In addition, they are a simple and convenient mechanism for exploring a volume by controlling the position and size of the preview blob. Finally, they are also a powerful UI for precisely controlling and steering a region-grow blob. Surface brushes, on the other hand, are somewhat data dependent and therefore noise sensitive as they slide around on data iso-surfaces. One is often forced to switch back and forth between sliding on data iso-surfaces or a painting plane depending on the selection scenario. When to perform this mode switching is often not apparent. The screen brush is a simple and effective selection tool for many selection scenarios. However, there are also many selection scenarios where it is difficult or impossible to find a view that allows for the separation of target regions and surrounding regions. In addition, the screen brush is best suited for selection only and unlike the blob tool, is not amenable to volume exploration due to its lack of depth control and its heavy dependence on volume rotation.

The experiments performed in this thesis attempt to demonstrate the effectiveness of a system for generating effective contextual views by providing multiple, intuitive ROI selection tools that can be combined in a serial fashion. The tools are coherently based around the use of a super-ellipsoid and a single unifying widget-based interface. The blend-able super-ellipsoid blobs complement a transfer function based UI in a volume rendering system by supporting separate, and potentially simpler, TFs for the region inside the blobs and for the context region outside. Feedback from participants is promising, and further improvements and additional capabilities can be made to increase the usability and flexibility of the system.

5.2 Future Work

5.2.1 Improved Widget Handles

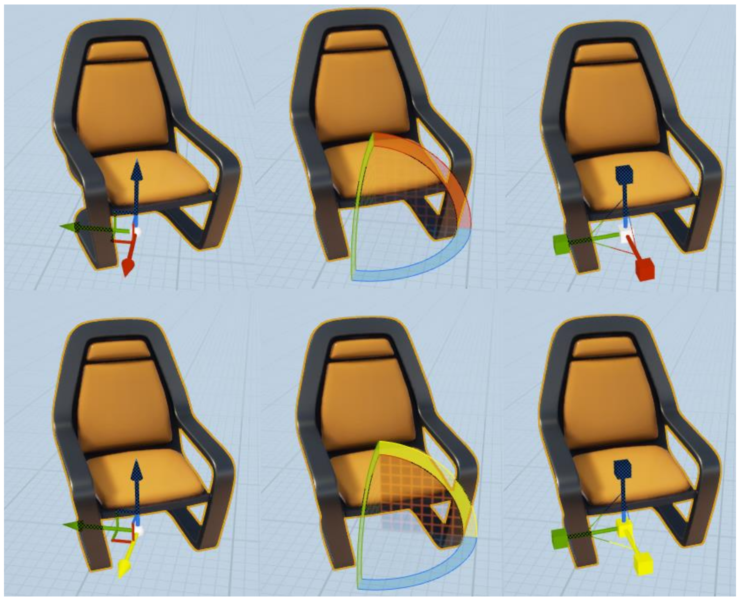


Figure 62: *The translation, rotation, and scale widget handles provided in Unreal Engine 4.*

The user study in Chapter 4 suggested that the widget handles weren't providing enough ease of use, positioning control and visual depth cue and this may have negatively impacted the performance of the blob tool. In particular, users had trouble positioning the blob in depth and rotating it. Game engine and 3D modelling software designers have had many years of experience and design iterations to address a similar object positioning problem in 3D views. For example, the transformation widget used in Unreal Engine 4 (Figure 62) is a possible replacement to the widget handle design in this thesis, especially for blob rotation. The rotation handle suggests its operation and the required mouse movements are easy to perform. In general, the advantages of a widget-based UI, demonstrated in part by its widespread and continued use in virtually all 3D modeling and game engine software, warrant further investigation into alternative handle designs.

5.2.2 Transfer Function Editor

Currently our system uses two sliders to determine the minimum and maximum visible voxel intensity range. These two sliders represent a simple step function in the alpha channel, with anything outside the range set to 0 and everything within it set to 1. While the simplicity of our current implementation allows users to quickly create contextual views, it also limits the flexibility of our system. Although potentially time consuming, the ability to create custom transfer functions gives users the ability to finely tune voxel visibility rules to handle any situation and potentially improve selection algorithm performance. For ideas, the transfer function editors of other popular medical volume rendering software like 3D slicer, ImageVis3D, and LiveVolume were evaluated. The transfer function editor used in LiveVolume was chosen for its simple and intuitive design and future implementations of our transfer function editor will emulate its design.

5.2.3 Accumulation Grid Super-Ellipsoid Field

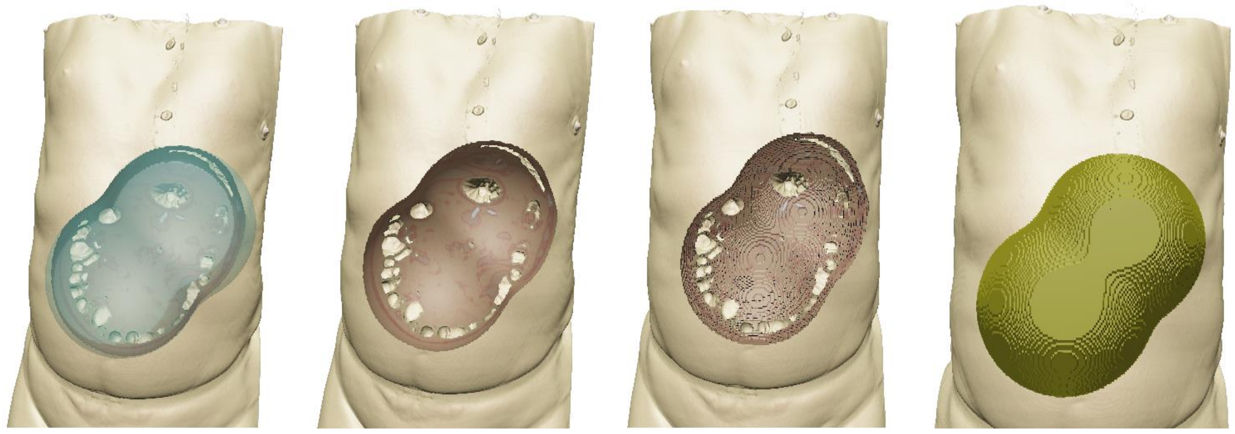


Figure 63: *An illustration of using the gradient of the blob field values for normals versus using the gradient of grid values in the accumulation grid. The left two images show the normals produced at the boundaries of a blob. The right two images show the boundary normals produced by the values in the accumulation grid.*

One issue with the accumulation grid is that the super-ellipsoid parameters and associated parametric equations that define a ROI are not currently utilized to compute an accurate normal vector for voxels on the boundary of a selected ROI. In the current implementation, normal vectors for these boundary voxels are roughly approximated from the binary values of the accumulation grid and results in the blocky appearance shown in the right most images in Figure 63. However, each voxel within the accumulation grid also stores an ID of the blob (or blended blobs) associated with it. In future work, to conserve the normals shown on the left most images in Figure 63, the blob parameters associated with an ID will be used to calculate accurate normal vectors for voxels inside an object (i.e. not surface voxels) that are on blob boundaries.

References

[1] P. Ljung, J. Kruger, E. Groller, M. Hadwiger, C. Hansen and A. Ynnerman, "State of the Art in Transfer Functions for Direct Volume Rendering," *Computer Graphics Forum*, vol. 35, pp. 669-691, 2016.

[2] F. Heckel, O. Konrad, H. Hahn and H. Peitgen, "Interactive 3d medical image segmentation with energy-minimizing implicit functions," *Computers & Graphics*, vol. 35, no. 2, pp. 275-287, 2011.

[3] G. Hamarneh, J. Yang, C. Mcintosh and M. Langille, "3D live-wire-based semi-automatic segmentation of medical images," in *SPIE Medical Imaging: Image Processing 5747*, 2005.

[4] M. Aliroteh and T. McInerney, "Sketchsurfaces: Sketch-line initialized deformable surfaces for efficient and controllable interactive 3d medical image segmentation," in *Advances in Visual Computing. ISVC07. Lecture Notes in Computer Science*, 2007.

[5] L. Faynshteyn and T. McInerney, "Context-preserving volumetric data set exploration using a 3d painting metaphor," in *Advances in Visual Computing. ISVC 2012. Lecture Notes in Computer Science*, 2012.

[6] G. Bonneau, T. Ertl and G. Nielson, "Generalizing Focus+Context Visualization," in *Scientific visualization: The visual extraction of knowledge from data*, G. Bonneau, T. Ertl and G. M. Nielson, Eds., Springer, 2006, pp. 305-327.

[7] H. Chen, F. Samavati and M. Sousa, "GPU-based point radiation for interactive volume sculpting and segmentation," *Visual Computer*, vol. 24, no. 7, pp. 689-698, 2008.

[8] "LiveVolume," [Online]. Available: www.livevolume.com. [Accessed 1 june 2017].

[9] Developers, The community of Slicer, "3DSlicer," [Online]. Available: www.slicer.org. [Accessed 2017].

[10] M. Hadwiger, P. Ljung, C. R. Salama and T. Ropinski, "Advanced Illumination Techniques for GPU volume raycasting," in *ACM Siggraph Asia 2008 courses*, Singapore, 2008.

[11] K. Engel, M. Kraus and T. Ertl, "High-quality pre-integrated volume rendering using harware-accelerated pixel shading," in *ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware - HWWS'01*, 2001.

[12] Computing, NIH/NIGMS Center for Integrative Biomedical, "ImageVis3D," [Online]. Available: http://www.sci.utah.edu/cibc/software/41-imagevis3d.html. [Accessed 24 November 2011].

[13] Pixmeo, "OsiriX DICOM Image Library," [Online]. Available: http://www.osirix-viewer.com. [Accessed 24 November 2011].

[14] S. Song, Y. Zheng and Y. He, "A review of Methods for Bias Correction in Medical Images," *Biomedical Engineering Review,* vol. 1, no. 1, 2017.

[15] J. Kniss, G. Kindlmann and C. Hansen, "Multidimensional transfer functions for interactive volume rendering," *IEEE Transactions on Visualization and Computer Graphics,* vol. 8, no. 3, pp. 270-285, 2002.

[16] G. Kindlmann, R. Whitaker, T. Tasdizen and T. Moller, "Curvature-based transfer functions for direct volume rendering: methods and applications," in *14th IEEE Conference on Visualization (VIS'03),* Seattle, WA, USA, 2003.

[17] M. Haidacher, D. Patel, S. Bruckner, A. Kanitsar and M. E. Groller, "Volume visualization based on statistical transfer-function spaces," in *Pacific Visualization Symposium (PacificVis'10),* Taipei, 2010.

[18] C. Correa and M. Kwan-Liu, "Size-based Transfer Functions: A New Volume Exploration Technique," *IEEE Transactions on Visualization and Computer Graphics,* vol. 14, no. 6, pp. 1380-1387, 2008.

[19] F. Y. Tzeng and K. L. Ma, "A cluster-space visual interface for arbitrary dimensional classification of volume data," in *Eurographics VGTC Symposium on Visualization (VisSim'04),* Norkoping, Sweden, 2004.

[20] F. Y. Tzeng, E. B. Lum and K. L. Ma, "An intelligent system approach to higher-dimensional classification of volume data," *IEEE Transactions on Visualization and Computer Graphics,* vol. 11, no. 3, pp. 273-284, 2005.

[21] J. Diepstraten, D. Weiskopf and T. Ertl, "Transparency in Interactive Technical Illustrations," *Computer Graphics Forum,* vol. 21, no. 3, pp. 317-325, 2002.

[22] J. Kruger, J. Schneider and R. Westermann, "ClearView: An Interactive Context Preserving Hotspot Visualization Technique," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 941-948, 2006.

[23] S. Bruckner, S. Grimm, A. Kanitsar and M. E. Gröller, "Illustrative Context-Preserving Exploration of Volume Data," *IEEE Transactiuons on Visualization and Computer Graphics,* vol. 12, no. 6, pp. 1559-1569, 2006.

[24] C. Correa and K. Ma, "The occlusion spectrum for volume classification and visualization," *IEEE Transactions on Visualization and Computer Graphics,* vol. 15, no. 6, pp. 1465-1472, 2009.

[25] J. Zhou, A. Döring and K. Tönnies, "Distance Based Enhancement for Focal Region Based Volume Rendering," in *Bildverarbeitung für die Medizin 2004*, Berlin, 2004.

[26] I. Viola and E. Gröller, "Smart Visibility in Visualization," in *First Eurographics Conference on Computational Aesthetics in Graphics, Visualization and Imaging*, Girona, Spain, 2005.

[27] E. Monclus, J. Dıaz, I. Navazo and P. Vazquez, "The virtual magic lantern: an interaction metaphor for enhanced medical data inspection," in *The 16th ACM Symposium on Virtual Reality Software and technology*, Kyoto, Japan, 2009.

[28] Y. Luo, J. Guitián, E. Gobbetti and F. Marton, "Context preserving focal probes for exploration of volumetric medical datasets," in *2009 International Conference on Modelling the Physiological Human (3DPH'09)*, 2009.

[29] T. Ropinski, F. Steinicke and K. Hinrichs, "Tentative results in focus-based medical volume visualization," in *5th International Symposium on Smart Graphics (SG06)*, 2005.

[30] A. Tappenbeck, B. Preim and V. Dicken, "Distance-based transfer function design: Specification methods and applications," in *Simulation and Visualization (SimVis06)*, 2006.

[31] C. Kirmizibayrak, "Interactive Volume Visualization and Editing Methods for Surgical Applications," Washington, DC, 2005.

[32] M. Burns, M. Haidacher, W. Wein, I. Viola and E. Gröller, "Feature emphasis and contextual cutaways for multimodal medical visualization," in *9th Joint Eurographics/ IEEE VGTC conference on Visualization (EUROVIS'07),* Aire-la-Ville, Switzerland, 2007.

[33] S. Bruckner and M. Gröller, "Volumeshop: an interactive system for direct volume illustration," in *16th IEEE Conference on Visualization (VIS'05),* Baltimore, MD, 2005.

[34] N. Radeva, L. Levy and J. Hahn, "Generalized Temoral Focus+Context Framework for Improved Medical Data Exploration," *Journal of Digital Imaging,* vol. 27, pp. 207-219, 2014.

[35] A. Norouzi, M. Rahim, A. Altameem, T. Saba, A. Ehsani Rad, A. Rehman and M. Uddin, "Medical Image Segmentation Methods, Algorithms and Applications," *IETE Technical Review,* vol. 31, pp. 199-213, 2014.

[36] T. McInerney and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Medical Image Analysis,* vol. 1, no. 2, pp. 91-108, 1996.

[37] E. Smistad, T. Falch, M. Bozorgi, A. Elster and F. Lindseth, "Medical image segmentation on GPUs - A comprehensive review," *Medical Image Analysis,* vol. 20, no. 1, pp. 1-18, 2015.

[38] S. Owada, F. Nielsen and T. Igarashi, "Volume catcher," in *2005 Symposium on Interactive 3D Graphics and Games (I3D'05),* New York, NY, 2005.

[39] G. Pintilie and T. McInerney, "Interactive Cutting of the Skull for Craniofacial Surgical Planning," in *IASTED Biomedical Engineering (BioMed2003),* 2003.

[40] L. Yu, K. Efstathiou, P. Isenberg and I. T, "Efficient structure-aware selection techniques for 3D point cloud visualization with 2DOF input," *IEEE Transactions on Visualization and Computer Grpahics,* vol. 18, no. 12, pp. 2245-2254, 2012.

[41] G. Shan, M. Xie, Y. Gao and X. Chi, "Interactive visual exploration of halos in large-scale cosmology simulation," *Journal of Visualization,* vol. 17, no. 3, pp. 145-156, 2014.

[42] A. Falcão and J. Udupa, "A 3D generalization of user-steered live wire segmentation," *Medical Image Analysis,* vol. 4, no. 4, pp. 389-402, 2000.

[43] T. McInerney and Y. Shih, "Sketch-line interactions for 3d image visualization," in *Advances in Visual Computing, Lecture Notes in Computer Science*, 2012.

[44] T. Igarashi, S. Matsuoka and H. Tanaka, "Teddy: a sketching interface for freeform design," in *ACM SIGGRAPH 2007 courses*, New York, 2007.

[45] D. Weiskopf, K. Engel and T. Ertl, "Interactive clipping techniques for texture-based volume visualization and volume shading," *IEEE Transactions on Visualization and Computer Graphics,* vol. 9, no. 3, pp. 298-312, 2003.

[46] R. Huff, C. Dietrich, L. Nedel, C. Freitas, J. Comba and S. Olabarriaga, "Erasing, digging and clipping in volumetric datasets with one or two hands," in *Virtual Reality Continuum and its Applications (VRCIA)*, 2006.

[47] O. Bernhard, B. Preim and A. Littmann, "Virtual resection with a deformable cutting plane," in *Proceedings of Simulation und Visualisierung*, Magdeburg, Germany, 2004.

[48] M. McGuffin, L. Tancau and R. Balakrishnan, "Using deformations for browsing volumetric data," in *IEEE Visualization*, 2003.

[49] C. Correa, D. Silver and M. Chen, "Constrained illustrative volume deformation," *Computer Graphics,* vol. 34, no. 4, pp. 370-377, 2010.

[50] A. Birkeland and I. Viola, "View-dependent peel-away visualization for volumetric data," in *Spring conference on computer graphics (SCCG)*, 2009.

[51] V. Soltészova, M. Termeer and M. Gröller, "Advanced volume painting with game controllers," in *25th Spring Conference on Computer Graphics (SCCG'09)*, Budmerice, Slovakia, 2009.

[52] H. Guo and X. Yuan, "Local WYSIWYG volume visualization," in *IEEE Pacific Visualization Symposium (PacificVis 2013)*, 2013.

[53] H. Chen, F. Samavati, M. Sousa and J. Mitchell, "Sketch-based volumetric seeded region growing," in *Third Eurographics conference on Sketch-based Interfaces and Modeling (SBM'06)*, Aire-la-ville, Switzerland, 2006.

[54] K. Shoemake, "Arcball: a user interface for specifying three-dimensional orientation using a mouse," in *Graphics Interface (GI 92)*, 1992.

[55] D. Norman, The Design of Everyday Things: Revised and Expanded Edition, Basic Books, 2013.

[56] G. Wyvill, C. McPheeters and B. Wyvill, "Data structure for soft objects," *The Visual Computer,* vol. 2, no. 4, pp. 227-234, 1986.

[57] M. Tigges, M. Carpendale and B. Wyvill, "Generalized distance metrics for implicit surface modelling," in *Tenth Western Computer Graphics Symposium*, 1999.

[58] A. Ricci, "A constructive geometry for computer graphics," *The Computer Journal,* vol. 16, no. 2, pp. 157-160, 1973.

[59] L. Yu, K. Efstathiou, P. Isenberg and T. Isenberg, "CAST: Effective and Efficient User Interaction for Context-Aware Selection in 3D Particle Clouds," *IEEE Trans. Vis. Comput. Graph.,* vol. 22, no. 1, pp. 886-895, 2016.

[60] American Psychological Association, "Publication manual of the American Psychological Association (6th ed.)," APA, Washington, DC, 2010.

[61] S. Zachow, E. Gladilin, R. Sader and H. Zeilhofer, "Draw and cut: intuitive 3D osteotomy planning on polygonal bone models," in *Computer Assisted Radiology and Surgery*, 2003.

[62] M. Christie, P. Olivier and J. Normand, "Camera control in computer graphics," *Computer Graphics Forum*, vol. 27, pp. 2197-2218, 2008.