Methods for Generating Volume Rendered Views from Registered Camera Images and Three-Dimensional Anatomical Data Sets

by Edward Michael Wakid

B.S. in Computer Science, May 2003, University of Maryland, College Park
M.S. in Computer Science, December 2005, The George Washington University

A Dissertation submitted to

the Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Science

August 31, 2011

Dissertation directed by

James K. Hahn
Professor of Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Edward Michael Wakid has passed the Final Examination for the degree of Doctor of Science as of August 31, 2011. This is the final and approved form of the dissertation.

METHODS FOR GENERATING VOLUME RENDERED VIEWS FROM REGISTERED CAMERA IMAGES AND THREE-DIMENSIONAL ANATOMICAL DATA SETS

Edward Michael Wakid

Dissertation Research Committee:

James K. Hahn, Professor, Advisor, Dissertation Director

Abdou Youssef, Professor, Chairman, Committee Member

Gabriel A. Parmer, Assistant Professor, Committee Member

Gabe Sibley, Assistant Professor, Committee Member

Morton H. Friedman, Research Professor, Committee Member

Acknowledgements

This page is to formally thank everyone who helped and supported me throughout the course of my research. Without these people I would have never been able to complete my dissertation and degree.

First, I would like to express my deepest gratitude to my advisor Professor James K. Hahn for his guidance and support throughout the years. Furthermore, for providing me with the opportunity to work in an atmosphere that allowed me to expand my knowledge in a new field and develop as a researcher. I would also like to thank Dr. Can Kirmizibayrak and Dr. Yeny Yim, whom I spent countless hours with, for their constructive feedback and suggestions. These two made it a pleasure to come to the lab every day.

I would like to thank my committee members, Professors Abdou Youssef, Gabe Parmer, Gabe Sibley, and Mort Friedman for their insightful feedback and assistance throughout the dissertation process.

Finally, I would like to thank my parents, brother, and wife for believing in me and sticking by my side throughout the entire process. I would also like to thank all of my friends and extended family for their patience throughout the years. Without your untiring support none of this would have been possible.

Abstract

This dissertation presents methods to generate high-quality volume rendered views from registered camera images and three-dimensional anatomical data sets. While camera images provide high-resolution color and texture information of a surface that belongs to a patient, anatomical data sets provide the structural information and reveal internal anatomy that cannot be seen directly. The objective of this research is to provide real-time visualization that displays the link between surface views of a patient's anatomy and the internal structures that lie behind so the correlation of information in both image modalities can be easily understood. In order to do this, a hybrid method combining 3D surface mesh parameterization and volume rendering is introduced, allowing image detail to be preserved independent of the volume's resolution. The visualization is provided by the viewpoint of a virtual camera, enabling the user to view the merged data from arbitrary viewpoints, where the depth of structures can be perceived through motion parallax and the context of the surrounding anatomy. This approach improves on previous methods by retaining volumetric information so the interior anatomy can be visualized. It also utilizes techniques tailored to modern graphics hardware to be fast and robust for a variety of medical applications. These proposed methods can be integrated into most medical systems that register camera images with 3D patient data and are flexible to be augmented by additional visualization techniques. Results show that we can successfully map multiple views onto volumes and provide high-quality volume renderings in real-time.

Table of Contents

Acknowledgements

This page is to formally thank everyone who helped and supported me throughout the course of my research. Without these people I would have never been able to complete my dissertation and degree.

First, I would like to express my deepest gratitude to my advisor Professor James K. Hahn for his guidance and support throughout the years. Furthermore, for providing me with the opportunity to work in an atmosphere that allowed me to expand my knowledge in a new field and develop as a researcher. I would also like to thank Dr. Can Kirmizibayrak and Dr. Yeny Yim, whom I spent countless hours with, for their constructive feedback and suggestions. These two made it a pleasure to come to the lab every day.

I would like to thank my committee members, Professors Abdou Youssef, Gabe Parmer, Gabe Sibley, and Mort Friedman for their insightful feedback and assistance throughout the dissertation process.

Finally, I would like to thank my parents, brother, and wife for believing in me and sticking by my side throughout the entire process. I would also like to thank all of my friends and extended family for their patience throughout the years. Without your untiring support none of this would have been possible.

List of Figures

Chapter 1 – Introduction

In the clinical setting, three-dimensional (3D) imaging modalities such as computed tomography (CT) and magnetic resonance (MR) have become ubiquitous for the assessment and diagnosis of patients. Two-dimensional (2D) color images, such as photographs and video, are frequently used with these modalities throughout the workflow of a variety of medical applications, including maxillofacial and endoscopic procedures. While photographs and video provide high-resolution color and texture information of patient surfaces, 3D anatomical data sets provide the topography for these surfaces and reveal structures inside that cannot be seen directly. Successful treatment comes in part by the ability to identify the correlation between these two sources of information.

An issue with current medical data visualization is that it is still performed, in many cases, by viewing three-dimensional data as two-dimensional slices, which requires a difficult 3D mental reconstruction of the patient. This type of assessment is highly subjective and is experience dependent. Furthermore, the problem is amplified in cases where multiple imaging modalities are used since there is typically no direct link between the image data. Image registration is the process of transforming these modalities into a common space where the correlation of data is identified. However, once the data has been registered a key problem is determining how to visually present the results in a way that is easy to understand.

Recent advances in graphics hardware have made it possible to render high-quality images for reasonably sized data sets in real-time, significantly improving the way data

can be visualized on commodity personal computers. This motivates new methods for data visualization that do not necessarily deviate from the standard clinical workflow or require specialized hardware. A 3D image fusion between the color image detail and 3D anatomical data provides realistic visualization, exploration, localization, and assessment of an electronic patient in the real world. Research has shown that these tools can improve the visual perception of information, remove ambiguity, reduce mental workload, and ultimately improve patient treatment.

While the clinical use of these technologies remains limited, the importance of developing such visualization tools is evident for next generation medical systems. The objective of this work is to identify the computer graphics issues associated with creating these visualizations, and develop methods to solve these problems. The goal is to obtain high-quality volume rendered views from the registered data for interactive 3D visualization in a variety of medical applications.

1.1 Motivation

The following describes two medical applications in which photographs or video are used and registered with 3D anatomical data, thereby motivating the presented work. However, it should be emphasized that these do not represent an exhaustive list of applications for this research.

Correctly planning and simulating surgical outcome is critical in facial surgery. Because of this there has been a growing interest in the generation of 3D virtual patient models using images acquired of the patient to improve patient care. Computer simulated predictions guide the treatment to the desired result, give the patient a reasonable preview

of the outcome, and serve as a communication tool between the orthodontist, surgeon, and patient [1]. Multimodal image visualization (e.g. combination of photographs, CT, and MR) provides surgeons with a tool where they can simulate and plan the surgery in a virtual environment. However the lack of realistic visualizations, real-time solutions, and user-friendly systems has remained a limiting factor.

With the advancement of minimally-invasive techniques and procedures, there is an increasing need for the development of improved visualization methods to resolve the limitations of using endoscopic devices. This is because the field of view provided by the scope is limited, only allowing a small region to be seen, making it difficult to correlate what is seen in the video and the anatomical data. Furthermore, the users control over the motion of the scope requires some expertise and is often limited by the small channel it is inserted into. A fusion of the color and texture information in the endoscopic video with 3D medical data enables 3D visualization and arbitrary navigation of the patient data. This includes creating wider views of the scene that the endoscope does provide and the ability to identify the spatial relationship between data more easily.

1.2 Problem Domain

Several research efforts have been made to combine color images with 3D anatomical data. In general, there are two approaches. The first is augmented reality (AR) [2], in which computer generated imagery is integrated onto the user's view of the patient in real-time. The conventional method for AR in medicine is to superimpose selected information from 3D anatomical data onto the user's view of the patient, to visualize the location of the interior structures that would normally be hidden from view. However,

while these approaches have shown to improve understanding of the spatial relationship of data, they suffer from a number of limitations. For one, AR visualization is fixed to the viewpoint of the user. This is disadvantageous, particularly in endoscopic procedures, where the patient's anatomy is observed indirectly through an endoscopic device. Endoscopes have a narrow field of view making it difficult to determine where the scope is relative to its surroundings, and is further complicated when the view must be correlated with 3D images. There are also limited movements when using a scope including actions such as looking directly behind the current view that make it difficult to find a desired viewpoint. Because of this it is difficult to visualize the data and accurately perceive depth of overlaid objects.

The second approach, sometimes referred to as augmented virtuality (AV) [3], integrates real world objects into a virtual environment. In this case, the visualization is provided through the viewpoint of a virtual camera, which can be arbitrarily positioned and oriented by the user, allowing interactivity not available with augmented reality. Depth is ascertained by the users interaction with the object though motion parallax and the context of surrounding structures. Augmented virtuality provides additional benefits, such as the ability to store the virtual model after it has been generated, which are not possible using an augmented reality approach directly. However, mapping images onto virtual models for an AV approach introduces additional challenges that can be less straightforward than producing AR visualizations.

Texture mapping is a computer graphics technique often used to add image detail to 3D models. By extracting a mesh from the 3D patient data, color images can be easily mapped to its surface by assigning texture coordinates to the vertices. Although mesh

renderings can be effective for visualizing surfaces, they omit potentially useful information from the original volume data set. On the other hand, direct volume rendering methods can generate images from the entire 3D volume without defining any explicit geometry, and can provide high-quality visualizations. Unfortunately, there are no direct means to texture map voxel-based (3D analog of a pixel) objects. Image color can be stored into a spatial data structure, such as a uniform grid, which can be sampled along with the volume. However, memory requirements become prohibitive in order to preserve fine image details. Adaptive grids, such as octrees, can reduce this memory requirement to a degree, but provide less efficient access to data at high refinement levels. The challenges associated with combining mapping camera images onto the 3D anatomical data in this way are the basis of discussion throughout this dissertation.

1.3 Proposed Solution

The proposed solution is an augmented virtuality approach utilizing a combination of mesh-based parameterization with volume rendering.

Difficulties with placing images onto the volume directly will be addressed by parameterizing a bounding mesh that approximates the surface of the volume, based on the camera parameters from registered viewpoints. The camera images linked to these viewpoints can then be texture mapped onto corresponding regions of the mesh surface. Texture mapping issues caused by projecting images in this way are resolved using a series of tailored computed graphics techniques and the graphics processing unit (GPU).

The texture mapped mesh can then be combined with the original volume data by identifying the 3D positions of the colors mapped onto the mesh with respect to the

volume. Using an original GPU-based raycasting technique, these colors are composited with volume samples in the correct visibility order. Merging data in this way allows the volume to be enhanced with image detail independent of its resolution. Volume rendering performance will also be increased by incorporating acceleration techniques using the bounding mesh.

This approach provides the following benefits:

- Real-time high-quality 3D volume visualization.
- Data navigation and exploration to identity structures and the spatial relationship of information.
- Ability to store the merged data into a file to be assessed and analyzed at a later time.
- A flexible implementation that can be integrated into most visualization systems.

1.4 Original Contributions

This dissertation describes the work believed to be original and contributory in the following aspects.

The main contribution of this dissertation is combination of computer graphics techniques to create volume rendered views from registered camera images and 3D anatomical data. This approach preserves image detail independent from the volume resolution, provides interactive high-quality volume visualization, and runs in real-time. These methods consist of the following:

- A fast projective texture mapping algorithm for surface parameterization, with measures to quantify the texture mapping quality of images mapped onto a mesh.

A fast GPU-based implementation is also provided to quickly compute texture coordinates for mesh vertices for streaming images.

- A surface painting method that stores the color of texture mapped images into a texture atlas. This method improves the texture mapping result and is well-suited for applications which require a large number of images to be mapped onto a mesh surface.
- A novel raycasting method that is accelerated using a bounding mesh. This method is capable of merging the color texture mapped on this mesh with volume samples, which allows the spatial relationship of the camera images and internal structures to be visualized together in a single volume rendered view.

Portions of work described in this dissertation have already appeared in five peer-reviewed publications [4, 5, 6, 7, 8].

1.5 Document Organization

The remainder of this dissertation is organized as follows:

Chapter 2 reviews work related to the topics throughout this dissertation. First, medical applications merging camera images and 3D anatomical data sets are discussed. Second, existing methods in the graphics domain for rendering volume data and mapping of color onto 3D surfaces are reviewed in order to identify potential solutions for our problem.

Chapter 3 presents an overview of the proposed solution. This chapter also discusses an important preprocessing step to extract a surface mesh from the original volume used

to achieve our final visualization and the general process of registering camera images with 3D anatomical data.

Chapter 4 describes the projective texture mapping method used to parameterize the mesh from registered image pairs. A GPU-based implementation that leverages graphics hardware to increase mapping performance is also presented.

Chapter 5 introduces a surface painting technique to map an arbitrary number of images onto the mesh and improve the texture mapping results.

Chapter 6 describes the volume rendering method used to merge the color texture mapped onto the mesh with samples from the original patient's volume in the correct visibility order.

Chapter 7 presents visual results from the proposed methods and evaluates their performance.

Chapter 8 concludes the dissertation with a discussion of potential future work.

Chapter 2 – Related Work

This chapter reviews work related to the topics discussed throughout this dissertation. The topics are broadly separated into two categories. First, existing medical research utilizing camera images and 3D anatomical data are discussed (Sections 2.1-2.3). Second, rendering techniques for displaying volume data and methods for mapping color onto 3D surfaces are reviewed (Sections 2.4 and 2.5).

2.1 Mixed Reality

Milgram and Kushino define mixed reality as technologies that involve the merging of real and virtual worlds, somewhere along the virtuality continuum (Figure 2-1), which connects completely real environments to completely virtual ones [3]. In our domain, live endoscopy (which provides color images) is an example of a real environment consisting of no virtual components. Virtual endoscopy (i.e. rendered views from a CT scan) is an example of a virtual environment with no real components.

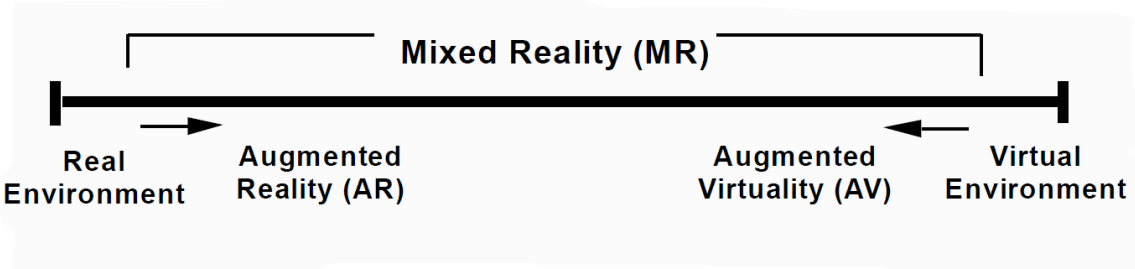


Figure 2-1: Reality-virtuality continuum (adapted from Milgram and Kushino [3]).

Augmented reality refers to views of the real environment that are augmented with virtual objects. The conventional approach for AR in medicine is to superimpose selected

information from the patient's 3D anatomical data set onto the user's view in order to visualize the location of the interior structures that would normally be hidden from view. An example of this is shown by Grimson et al. [9] in which brain images are registered and rendered onto a video view of the patient (Figure 2-2).

Figure 2-2: Visualization image of brain merged with video view [9].

On the other hand, augmented virtuality refers to cases where virtual environments are augmented with real world objects. Paul et al. [10] presented a system that created AV scenes for multimodal image-guided neurosurgery. Figure 2-3 shows an example of these scenes that include a 3D surface mesh of the operative field reconstructed from a pair of stereoscopic images acquired through surgical microscope, and 3D surfaces segmented from preoperative multimodal images of the patient.

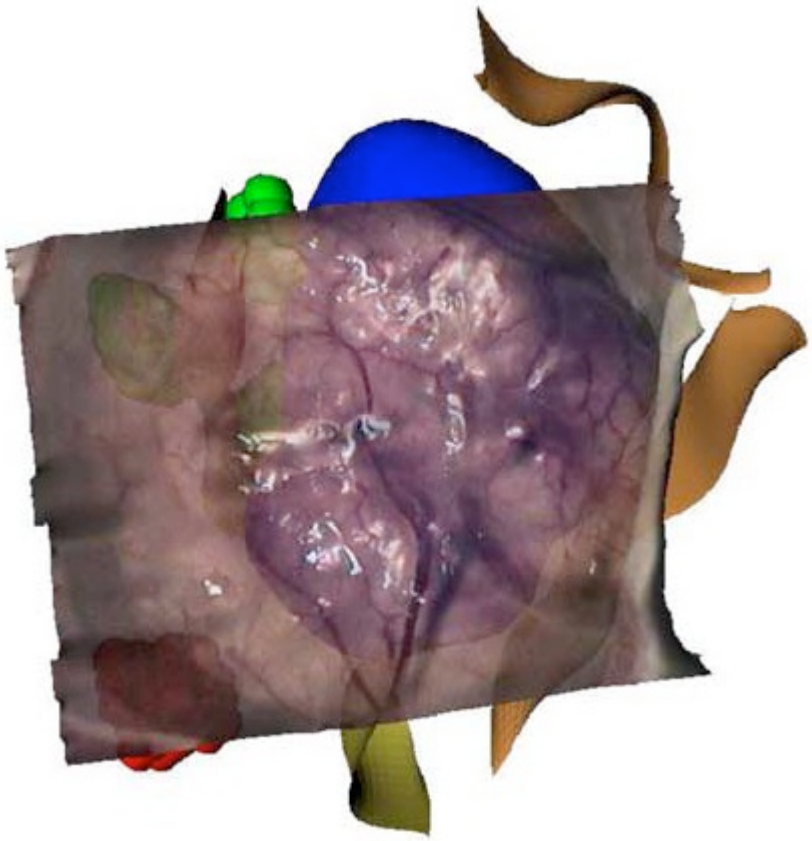


Figure 2-3: Augmented virtuality visualization of the operating field for image-guided neurosurgery [10].

This taxonomy is important for classifying the range of technologies using a combination of real and virtual components, such as camera images and three-dimensional anatomical data sets.

2.2 Facial Surgical Planning and Simulation

There have been several approaches to generate individualized 3D patient models for computer-based facial surgical planning and simulation. Ip and Yin [11] proposed a method to construct 3D head models from two photographs (a front and side view) and mapping them onto a 3D generic mesh which is deform to fit the extracted facial features.

[11, 12, 13, 14]. In some earlier works 3D laser scanning was used [15, 16], but was limited by slow acquisition speed, patient eye protection, and the inability to capture surface texture. Stereophotogrammetry has also been proposed to generate a surface of the face from converging pairs of views and texture mapped using images acquired by color cameras [17, 18], which can be viewed by the orthodontist from any angle (Figure 2-4).

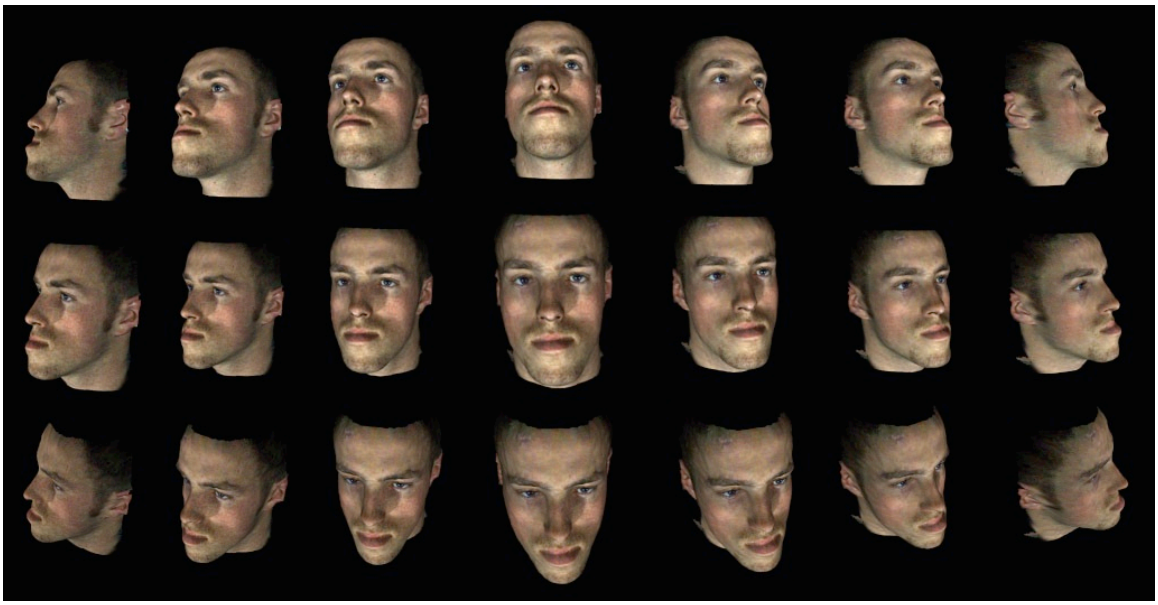


Figure 2-4: 3D imaging of the face enables the orthodontist to evaluate the face from any direction [18].

Many CT-based approaches have also been proposed. Xia et al. [19, 20] developed a surgical planning system for soft-tissue prediction based on an individualized texture-mapped facial model constructed from a generic mesh. Iwakiri et al. [21] derived a texture mapping method to place a set of orthogonal photographs onto a surface mesh obtained directly from the patient's CT scan. The result of their approach is shown in Figure 2-5.

Figure 2-5: Photographs (top) textured onto a 3D facial model derived from CT (bottom) [21].

In addition to published literature, a number of commercial software packages have emerged including 3dMDvultus (http://3dmd.com/), Dolphin 3D (http://www.dolphinimaging.com/), among others that are regularly used in practice to generate and visualize individualized 3D facial models. An example 3D patient model reconstructed from photographs and CT using the 3dMD system is shown by Figure 2-6.

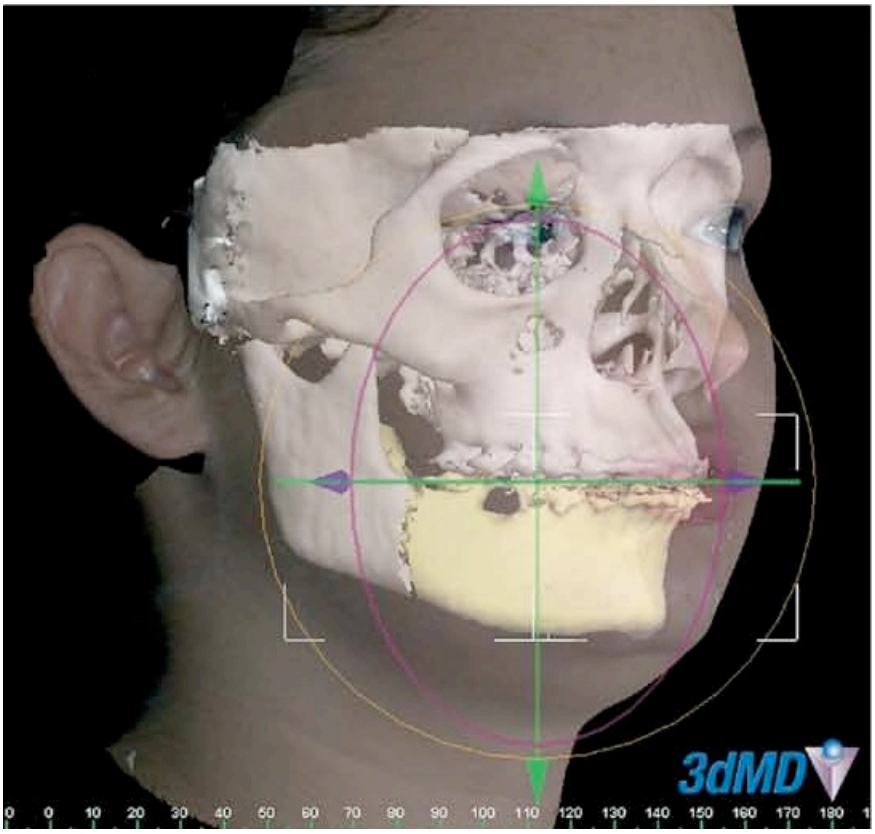


Figure 2-6: Patient-specific anatomical reconstruction using photographs and CT using the 3dMD system [22].

One issue with these approaches is they texture map surfaces of the patient's face and do not include volumetric views. Given that a patient volume data is available, additional segmentation is required to visualize internal structures. The Anatomage Invivo imaging software (http://www.anatomage.com) is an example of a newer system which can generate volume rendered views of a patient with a photograph fused with their CT, however the image must be visualized in front of the volume.

2.3 Endoscopic Surgery

Augmented reality has been successfully applied to image-guided bronchoscopy [23, 24] to assist in the removal of central-chest lymph nodes. Before the procedure, the physician constructs a list of predefined biopsy sites and generates a surface of the airway using the 3D CT of the patient. Two images are obtained, a real view of the airway surface seen through the bronchoscope and a virtual rendering of the same surface with visible target regions provided by the planning stage. Once the virtual image has been registered to the bronchoscope image, visible target locations seen in the virtual image can be overlaid onto the bronchoscope video, aiding the physician's ability to determine where lymph nodes are behind the airway walls (Figure 2-7).

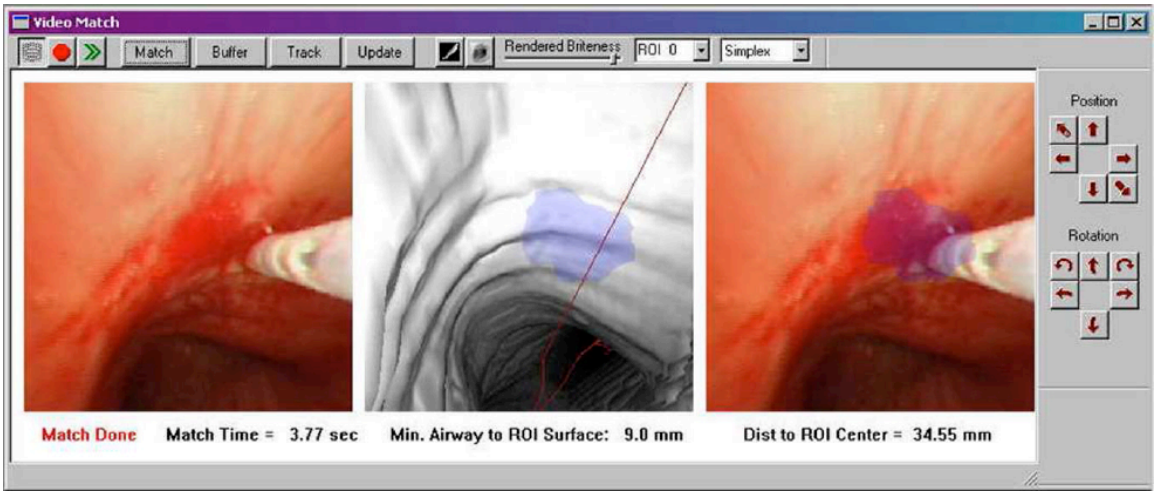


Figure 2-7: Image-guided bronchoscopy [24]. Bronchoscope image (left), registered virtual view with target nodule displayed (middle), and target nodule overlaid onto the bronchoscope image (right).

The problem with superimposing virtual information onto video is the loss of depth perception. When an object is augmented onto an image, it is difficult to determine how far into the scene it actually is. In the case where augmented reality is used to display objects underneath the surface, the virtual object can be perceived as floating above the

surface [25, 26]. Lerotic et al. [27] used a non-photorealistic rendering technique to improve the accuracy of users depth perception by providing see-through of the anatomical surface while accentuating its salient features over the virtual object (Figure 2-8).

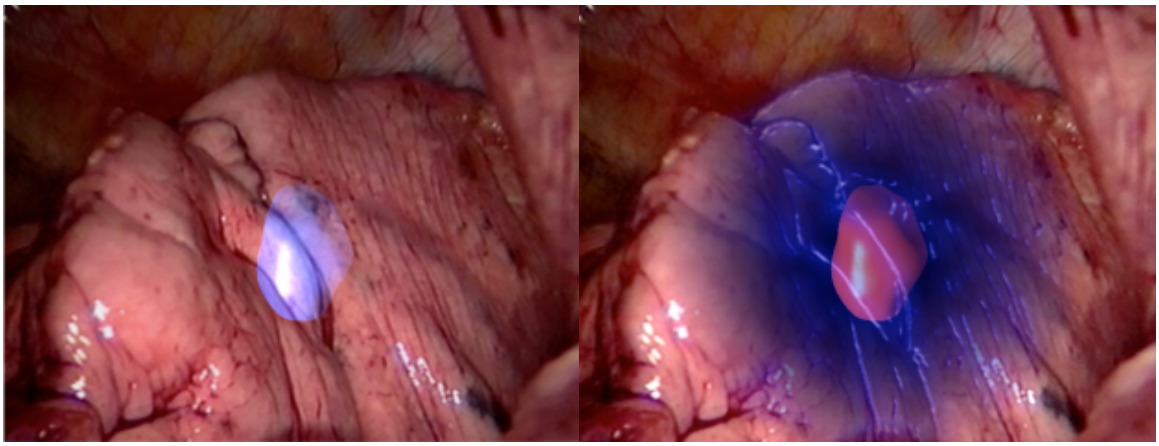


Figure 2-8: *pq*-space based non-photorealistic rendering for augmented reality [27]. Traditional augmented reality overlay (left) versus non-photorealistic augmented reality (right).

However, while the depth issue can be lessened to a degree, the image remains fixed to the viewpoint of the endoscope. Because the surgical field is observed indirectly, one of the problems tightly linked with endoscopic surgery is that the narrow field of view can make it difficult to locate objects seen through the endoscope [28]. Furthermore, the insertion of an endoscopic device into the patient generally results in reduced degrees of freedom with the movement, which makes scope navigation difficult.

In some commercial applications, such as QuickTime VR [29] and Surround Video [30], a wide field of view of an environment is created by stitching multiple photographs or video frames together to form visual panoramas. These types of mosaicing approaches have been adopted for endoscopic applications, stitching video frames together in order

to form a wider views, of an otherwise limited view, in effort to improve evaluation of the surgical or diagnostic scene [31, 32, 33, 34, 35, 36, 37].

Dey et al. [38, 39, 40] proposed creating panoramic mosaics from endoscopic images and surfaces for neurosurgery to enable stereoscopic visualization and 3D navigation of a virtual patient from arbitrary perspectives (Figure 2-9). Endoscopic video was texture mapped onto registered surfaces extracted from 3D preoperative data sets. Rai and Higgins [41] proposed mapping bronchoscopic video onto CT-based renderings to use 3D navigation for visually detecting early stages of lung cancer. Paul et al. [10] applied augmented virtuality to multimodal image-guided neurosurgery. Their method reconstructs and texture maps a 3D surface from a pair of stereoscopic microscope images, and renders them with target structures extracted from CT and MR data. Wang et al. [42] proposed an approach to texture map laparoscopic images onto pre-built 3D surface models of target anatomy derived from volumetric data sets using 3D-2D feature correspondences (Figure 2-10).

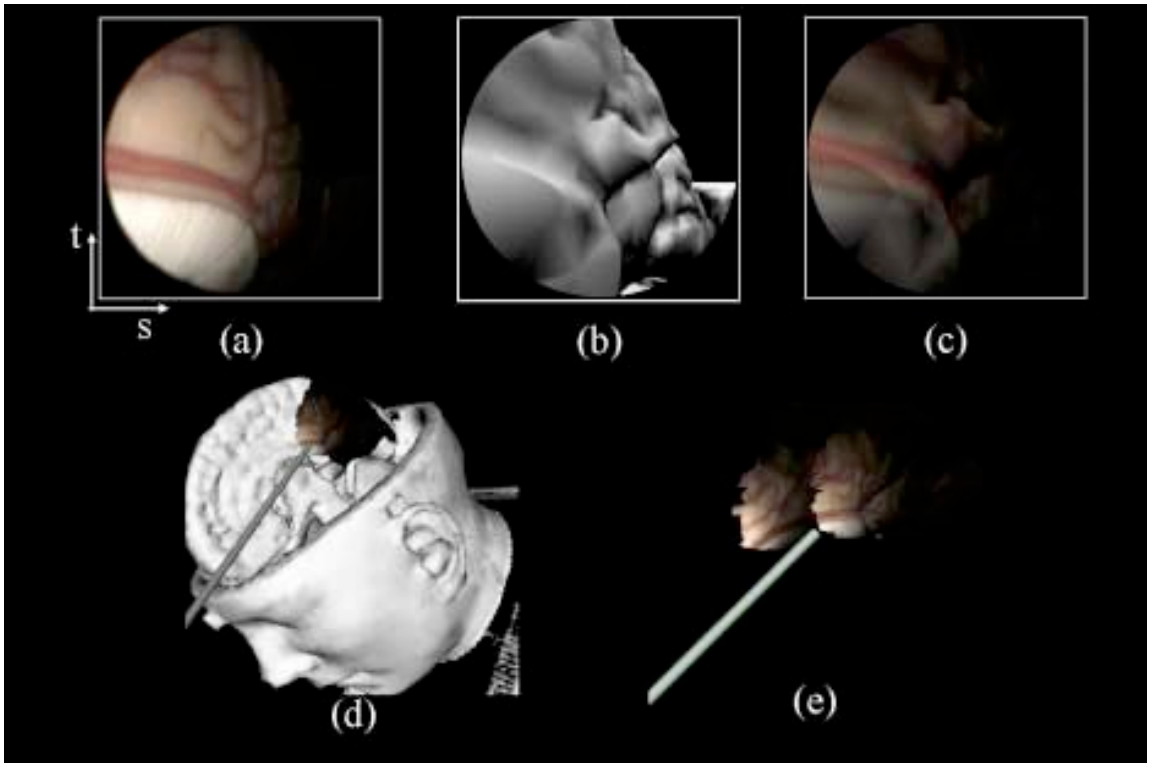


Figure 2-9: Texture mapping of multiple freehand endoscopic views [40]. (a) Acquired endoscopic image. (b) Virtual endoscopic image corresponding to (a). (c) Endoscopic image shown in (a) mapped to the surface, viewed through the virtual camera. (d) Texture-mapped surface viewed from another perspective. (e) Implementation of multiple endoscopic views with surface patches.

Figure 2-10: Endoscopic video of a pig liver texture mapped onto a 3D anatomical model [42]. Pig liver with initial video frame texture mapped after 3D-2D correspondences are made (a), more images mapped after minor camera movements (b), fully textured surface (c).

These approaches have the advantage that the user can create new views of the data, resolving many of the aforementioned issues with augmented reality approaches. With

the exception of the system developed by Paul et al [10], the objectives of these works were not focused on visualizing structures that lie behind the surface seen in the video, which is why surface-based representations are sufficient. Unfortunately, other means are necessary to display the data volumetrically. In the following sections we review volume rendering techniques and methods for parameterizing surfaces so image color can be placed onto them.

2.4 Volume Rendering Techniques

Volume rendering is considered an advanced technique for visualizing discretely sampled three-dimensional data sets. The two common approaches used to display volumetric data are surface extraction (e.g. isosurfacing) and direct volume rendering algorithms.

The marching cubes algorithm [43] is a popular technique for extracting surfaces from a 3D scalar field. An isovalue is used to define a surface from the volume and then geometric primitives are fit to the volume to reconstruct the 3D surface. A cube "marches" through the data, assigning each of the eight neighboring scalar values to one vertex of the cube and comparing them against the selected isovalue. Values greater than or equal to the isovalue are considered as inside or on the surface, and values less than the isovalue are considered outside the surface. Since there are two states, inside (1) or outside (0), and eight vertices, there are 2^8=256 possible polygon configurations for each cube which are accessed through a look up table. Once the surface has been constructed, conventional rendering techniques may be used.

Other methods perform a direct rendering of the isosurface from the volume without the need of polygonal meshes [44, 45]. However, the primary disadvantage of surface-based methods is that only data corresponding to the selected isovalue is available to be rendered while the remaining information is either discarded or not displayed.

Direct volume rendering techniques [46, 47] are used to display three-dimensional data without direct use of geometry. Contrary to surface-based rendering, volumetric approaches allow the data to be visualized as a whole, which is one of its most significant advantages. Direct volume rendering algorithms can be classified as image-order or object-order methods. While the following descriptions are not an exhaustive list, they represent some of the most well-known techniques.

Raycasting is an image-order approach for volume rendering. Rays are cast from the eye through each pixel in the image into the volume (Figure 2-11). The general idea is to directly evaluate the volume rendering integral for each ray. The volume is resampled at the discrete positions along the ray, and since rays emanate from the eye, compositing naturally occurs in front-to-back order. Most raycasting methods employ optimization from two observations of volumetric data by Levoy [48]. The first is based on the fact that there are often large regions of empty voxels that do not contribute to the final rendering. The limiting of ray traversal to non-empty regions is known as empty-space skipping. The second observation notes that once opacity along a ray has accumulated above a threshold, further samples are not needed and ray traversal may stop. This is known as early ray termination.

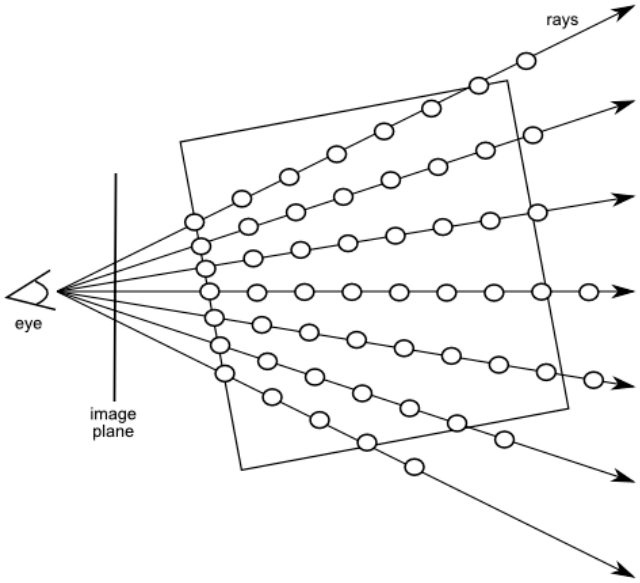


Figure 2-11: Raycasting principle. A ray is cast from the eye through each pixel in the display into the volume. The volume is sampled, accumulating color and opacity.

Texture slicing [49, 50] is an object-order approach where multiple 2D slices are used to sample the volume. Slices are located within the volume and can be aligned with respect to the object or to the viewing plane (Figure 2-12). Slices can be rendered back-to-front or front-to-back manner using appropriate compositing for that scheme. The appeal for texture slicing originates from texture mapping and blending being directly supported by graphics hardware, along with high-performance.

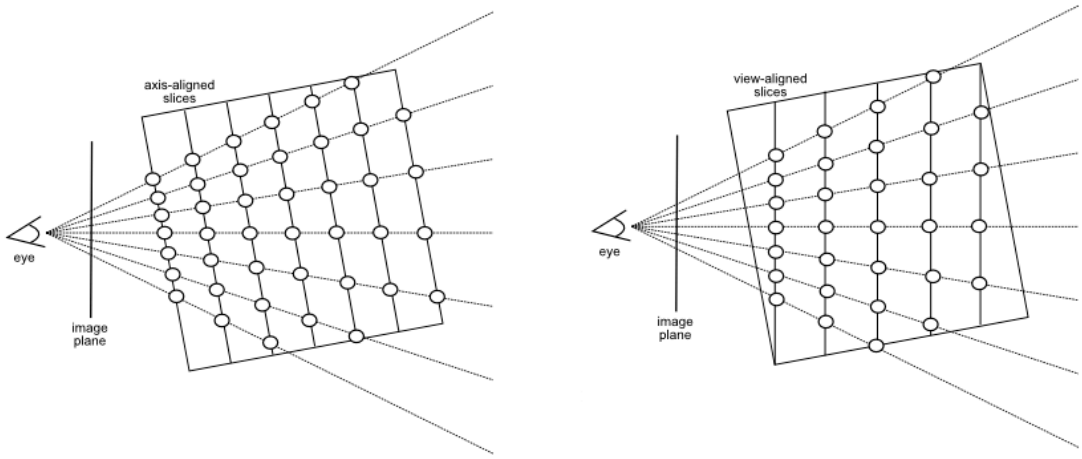


Figure 2-12: Sliced-based volume rendering. Axis-aligned slices (left) and view-aligned slices (right) are illustrated.

Until recently, texture slicing was the predominant technique for volume rendering. Raycasting has been another popular volume rendering technique, since it can produce high-quality images and is a flexible algorithm. However, CPU implementations of these approaches tended to be slow, and hardware optimizations were hindered by early GPU functionality. Modern graphics processors exhibit an intrinsic parallelism which makes raycasting a well-suited technique since ray calculations are performed independent from each other. The advances in graphics hardware and rendering techniques have made it possible for high-quality visualizations of dense volumetric data in real time.

Krüger and Westermann introduced one of the first GPU-based raycasting implementations [51]. Volume dimensions are scaled into the range of [0, 1] and a volume bounding box is created. Each vertex of the bounding box is assigned a RGB equal to its local coordinate position, which also conveniently serves as its 3D texture coordinates (Figure 2-13). The front and back faces of a volume bounding box are drawn into two separate color buffers and RGB values are interpolated across each face. The ray

traversing through each pixel can then be determined by accessing that pixel in both rendered images. The starting position of the ray is the value retrieved from the color of the front face image, and the length and direction is determined by subtracting the front face value from the back face value. Samples inside the volume bounds are found by incrementally moving the ray from the start position along its direction at a specified step size. The color and opacity of a pixel is accumulated until a threshold is reached or the ray exits the volume.

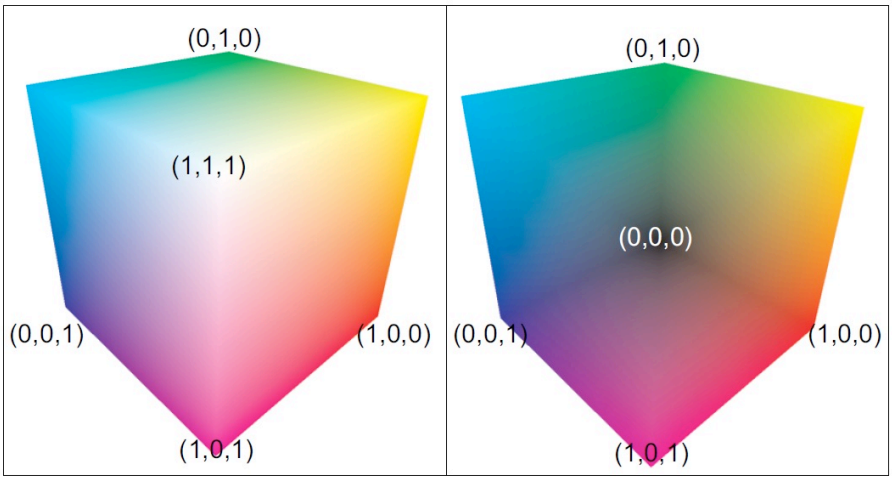


Figure 2-13: GPU-based raycasting [51]. Rendering front faces (left) and back faces (right) of the volume bounding box in order to generate ray directions and texture coordinates of first ray intersection points.

For large volumetric data sets, some raycasting methods utilize blocking schemes by decomposing the volume into smaller sub-volumes to facilitate empty-space skipping [44, 52]. Each block stores min-max values for a set of voxels, which are used to determine if that block is active (non-empty) or inactive (empty). By rendering the active blocks, rays can be started on their bounding faces and cast into the volume. Since blocks are culled on the CPU, each time the state of the blocks changes the data must be reloaded onto the GPU. Hong et al. [53] perform raycasting on each block individually by

sorting them into a correct visibility order and projecting layers. However there is an overhead incurred in the setup process.

The polygon assisted raycasting algorithm (PARC) [54] is a hardware-assisted method that encloses the volume using bounding geometry to generate ray positions that start and end close to the true surface. A similar implementation of this method was done more recently on the GPU [55], but it uses an isosurface mesh, such as one obtained by using marching cubes, for defining ray segments. This results in well-defined ray segments that minimize the amount of empty sampling outside the visible bounds of the volume. A disadvantage of this method is that it requires a new surface to be extracted each time the transfer function changes.

Alternative object-order approaches have been proposed. Mensmann et al. [56] use occlusion frustums as proxy geometry obtained from previously rendered frames. Vidal et al. [57] use a kd-tree to remove empty space in a preprocessing stage, however there is a large overhead incurred each time the transfer function changes. Liu et al. proposed an acceleration method for raycasting using proxy spheres [58]. Proxy spheres, in a sense, are splats which are used to bound active blocks instead of a cube. Bricking methods require 12 triangles per block versus the proxy sphere method which only requires one point primitive. The size of the point, which acts as a 2D projection of the proxy sphere, is controlled using a size function. As a result, the amount of geometry required is greatly reduced while providing tight fitting ray segments. The performance of this method depends on determining an optimal block size for fitting spheres to, which is dependent on the data set itself.

2.5 3D Surface Parameterization

Surface parameterization is at the core of texture mapping [59], a technique for adding image detail to computer generated models. It is ubiquitous in computer graphics and has been extensively studied over the past few decades. Texture maps are typically applied to polygons to place image information, the most common being color, across the face to add details to an otherwise simple surface. The widespread use of texture mapping is due to its ability to significantly enhance the appearance of an object at the expense of low computational cost. One of the fundamental challenges of texture mapping is the parameterization of the model in texture space, which can be very difficult for implicit surfaces and complex geometric objects. Further explanation can be found in Heckbert's survey on texture mapping [60]. Similarly, surface painting techniques are used to paint colors directly onto the vertices of 3D polygonal models. However, vertex coloring restricts the color variation to the surface resolution requiring an upsampled surface representation or alternative 3D data structure.

Bier and Sloan introduced a texture mapping technique for unparameterized surfaces that is performed in two parts [61]. Their method projects convex shapes onto an intermediate surface such as a cylinder, box, sphere, or plane which can be mathematically parameterized. Shibolet and Cohen-Or adopted this technique for voxel-based models to color internal cavities for virtual endoscopy [62]. Their method uses a dilation and process to map complex, non-convex surfaces into a convex shape before mapping onto a cylindrical surface. Unfortunately, this solution only works for short tube-like structures and does not provide the control necessary to accurately place arbitrarily located textures onto the surface. Shen and Willis also proposed an approach

for texture mapping volumetric models using a two-part mapping approach which projects the surface onto an intermediate shape from the center of the object [63]. The intermediate surface can be directly painted on or textured but requires some user intervention. Furthermore, these methods encounter many-to-one mappings where the projection maps multiple surface points to the same location. Therefore, multiple intermediate templates are necessary. In general, these types of mappings also have inherent image stretching distortions that do not make them ideal for this application.

An alternative approach is to paint the volume directly. Color is stored into a separate 3D texture that corresponds with voxels in the original volumetric data set and can be sampled accordingly. The portion of the 3D color texture written into must be reloaded into graphics memory each time the user paints on the object. This is typically prohibitive for real-time applications. Bürger et al. [64] introduced a volume editing technique that allows a user to interactively paint and edit a 3D color texture by using recent GPU functionality to perform fast write operations. While this technique provides an effective means for coloring volume data, a high resolution 3D texture would be required to store camera image detail with the majority of memory unused due to non-surface voxels. Additionally, their render to 3D texture method relies on a spherical brush, which does not trivially adapt for placing arbitrary images onto a surface.

Octree textures were first introduced by Benson and Davis [65] and DeBry et al. [66] to paint onto unparameterized surfaces and later implemented on the GPU by Lefebvre et al. [67] and Lefohn et al. [68]. Octree textures offer adaptive detail, regular sampling, and continuity across the surface. However, the performance is limited by dependent memory access operations to determine affected surface samples. Bürger et al. [69] proposed a

data structure for sample-based surface painting called the orthogonal fragment buffer but is subject to the typical limitations of resampling approaches, such as loss of detail caused by undersampling or blurring of sharp features. Filtering can also be significantly more expensive than in the 2D texture domain since adjacent samples may reside in different sampling grids and positions.

Chapter 3 – Merging Camera Images with 3D Anatomical Data

3.1 Overview

The proposed methods for generating volume rendered views from registered camera images and 3D anatomical data sets are based on tailoring techniques to combine polygonal texture mapping with volume raycasting. The general idea is to handle the problems associated with adding color detail to volumes by first parameterizing and texturing mapping an approximating bounding mesh, and then merge that result with the original volume. By doing so, performance and image detail are increased while reducing overall complexity. This is due in part to the support provided from graphics hardware. This resulting visualization for these methods possesses the following properties that are not simultaneously available with existing approaches:

1. The scene can be rendered from arbitrary viewpoints chosen by the user. As a result, visualization is not limited to the location of a real camera source, enabling 3D navigation and exploration.
2. The information present in the original patient volume is retained, including the interior anatomy that would typically be discarded with surface-based methods.
3. Texture mapped surfaces can be visualized semi-transparently, providing depth cues and the spatial relationships of underlying anatomical structures.
4. The texture mapping and raycasting processes are performed in real-time while providing high-quality volume rendered views.

There are five principle steps to do this, consisting of the following:

1. Mesh extraction

2. Registration
3. Mesh parameterization
4. Surface painting
5. Volume rendering and visualization

The remainder of this chapter focuses on steps 1 and 2, which addresses acquiring the surface mesh from the volume and the general process to identify the correspondence between camera images and the extracted surface.

3.2 Mesh Extraction

Isosurfacing is commonly employed data visualization method for three-dimensional scalar fields (e.g. CT/MR). An isosurface is a surface which represents points of constant value within a volume of space. 3D medical imaging modalities, such as CT and MR, consist of multiple two-dimensional image slices in which their pixel contents contain some form of anatomical information. For example, the pixel information for CT images are scalar values stored in Hounsfield units (HU), a quantitative scale for describing radiodensity, or the relative transparency of a material to X-rays and other radiation. Higher values indicate greater opacity to X-ray photons. These values typically range from -1024 to 3071 and pick up substances such as air, water, fat, muscle, skin, and bone. Using a selected isovalue (i.e. threshold of radiodensity), a mesh that corresponds to the surfaces in the volume that will be observed by the camera can be obtained through an isosurface extraction method, such as the marching cubes algorithm [43]. This does require the user to have knowledge of the surface in the volume.

Figure 3-1 shows an example of a mesh extracted from the skin boundary in a CT scan. This mesh provides a geometric approximation for a surface in the volume. The resulting mesh provides two functions. First, it provides a representation to parameterize the surface and texture map camera images onto (Chapters 4 and 5), and second, it accelerates ray traversal by minimizing the number of empty-space samples (Chapter 6). One the mesh has been obtained, it is necessary to register it to the images captured by the camera to identify the correspondences between data sets.

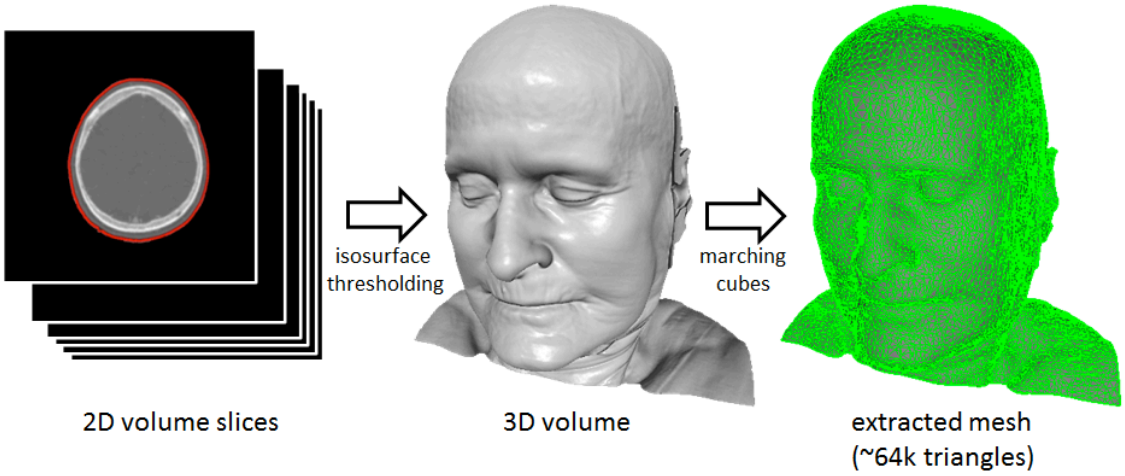


Figure 3-1: Mesh extraction process. Starting with a series of 2D volume slices (left), a 3D isosurface rendering (middle) of the volume is generated by interactively thresholding an isovalue. A surface mesh is extracted from the volume using this isovalue and the marching cubes algorithm.

3.3 3D-to-2D Registration

Before images can be mapped onto the surface mesh, it is first necessary to identify the relationship between 2D pixels in the camera images and their corresponding 3D position in the volume data set, a process known as image registration. The registration problem can be thought of as matching the viewpoints of two cameras, as shown by Figure 3-2. The first of which is a real camera (i.e. a physical camera in real world) that acquires 2D color images I_r of the patient. The second is a virtual camera that generates

2D rendered images I_v of the patient's 3D anatomical structures. While the two imaging sources do not consist of identical information, they provide different forms of the same physical 3D structures. As shown in Figure 3-2, there is considerable visual similarity between both images, which in turn makes registration possible.

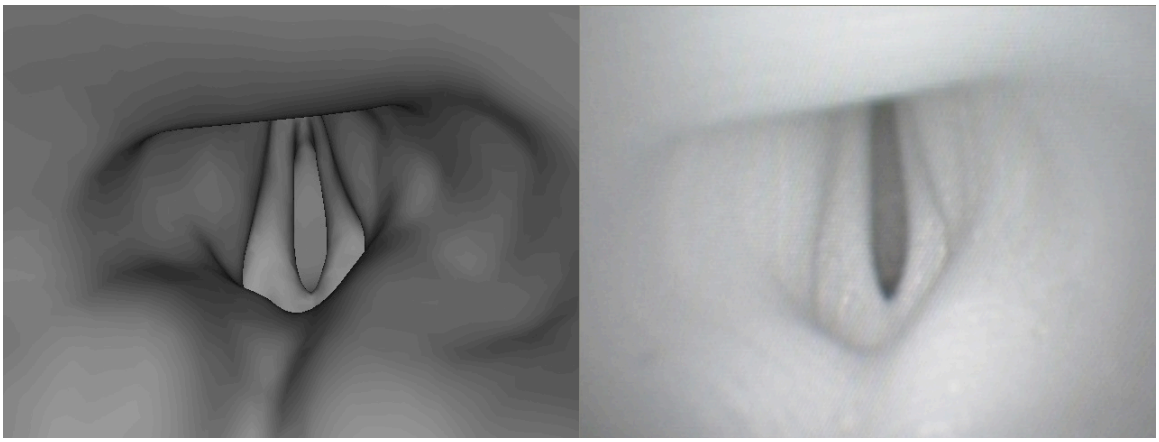


Figure 3-2: Example of a registered pair of CT-video images. On the left is a virtual mesh rendering I_v from a patient's 3D CT scan. On the right is an endoscopic video frame I_r, from a phantom patient, containing color and texture information.

The goal of the registration is to align the viewpoints of these two cameras such that if the two cameras are perfectly registered, a pixel (x_i, y_i) in the camera image I_c and rendered image I_v arise from the same physical 3D point. In order to align the two viewpoints, the real and virtual cameras must share the same imaging geometry. This is performed by determining the real camera's parameters (Section 3.4) and modeling the virtual camera from those parameters (Section 3.5). There is an abundance of literature regarding 3D-2D registration. Many of which include maximization of mutual information to align two images [70] and variations of the iterative closest point (ICP) algorithm [71] to align feature points in both spaces.

It is important to note here that registration can be very complex. For example, additional consideration may be necessary for differences in the real and virtual images such as variations in the patient's pose, deformable tissue, and anatomy that may be detected in one image but not in the other (e.g. hair). However, these considerations are outside the scope of this dissertation. This work focuses on the mapping and rendering issues once the data has been registered.

3.4 Camera Geometry

Each camera source can be modeled as a pinhole camera [72]. The pinhole camera model provides a convenient representation that describes the transformation of a 3D point in a scene to its 2D projection on the image plane. These transformations are constructed from intrinsic parameters, such as focal length and principal point, and extrinsic parameters corresponding to the position and orientation of the camera relative to the world coordinate frame.

3.4.1 Intrinsic Parameters

The intrinsic properties of a camera describe its internal characteristics and determine the projection of an object onto an image. Using the pinhole camera model, the projection of a 3D world point $(x_w, y_w, z_w)^T$ and its corresponding 2D point $(x_i, y_i)^T$ on the camera image plane is defined as:

$$x_i = f\frac{x_w}{z_w}, \quad y_i = f\frac{y_w}{z_w}, \tag{3.1}$$

where f is the camera focal length. For homogeneous coordinates, the above equations can be expressed in matrix notation:

$$\omega \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \tag{3.2}$$

where ω is a homogeneous scaling factor. This model assumes that the origin of the pixel coordinate system corresponds to the center of the image, or principal point. However, pixel coordinates are typically defined with an origin at the top left corner of the image with the positive x-axis pointing to the right and positive y-axis pointing down. The position of the principal point $(c_x, c_y)^T$ can be integrated into the perspective projection equation, which now becomes:

$$\omega \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}. \tag{3.3}$$

Camera parameters are geometrically illustrated by Figure 3-3 below.

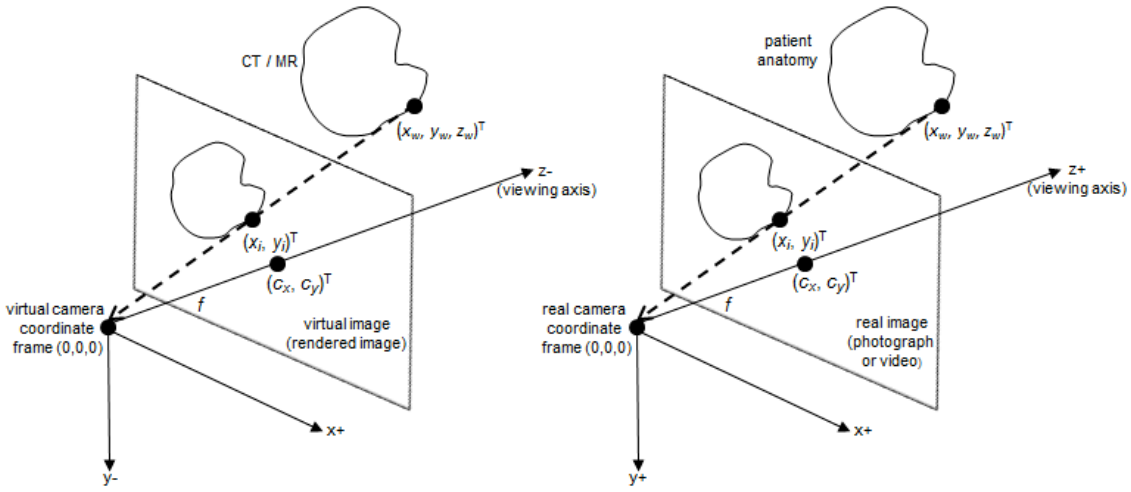


Figure 3-3: Camera imaging geometry.

These parameters can be obtained through a camera calibration, such as the one provided by Roger Tsai [72]. From the calibration, the camera focal length f, principal point (c_x, c_y), and radial lens distortion coefficient k are obtained. Lens distortion is generally undesirable and is more practical to remove than to incorporate into the virtual camera model. For these reasons camera images are corrected for this distortion as they are acquired using the parameter k.

3.4.2 Extrinsic Parameters

While the intrinsic parameters describe a camera's projection of 3D points to 2D, extrinsic parameters describe the camera's pose or viewpoint (i.e. position and orientation) relative to the world coordinate frame (Figure 3-4). The orientation of the camera is denoted by a 3 x 3 rotation matrix R and the position is denoted by a 3 x 1 translation vector T.

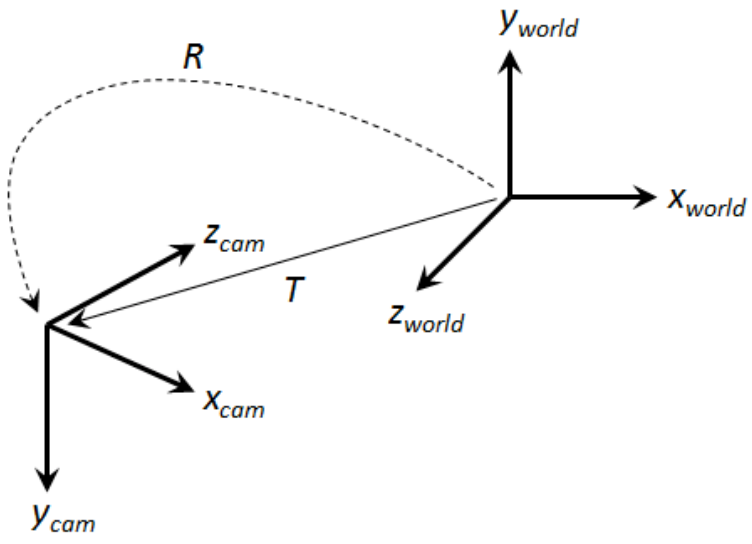


Figure 3-4: Transformation from the world coordinate frame to the camera coordinate frame, specified by a rotation matrix *R* and translation vector *T*.

Therefore a 3D world point $(x_w, y_w, z_w)^T$ must be also transformed relative to the camera coordinate frame prior to being projected on the 2D image plane as follows:

$$\omega \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.4}$$

Using these parameters, we can now define a virtual camera.

3.5 Modeling a Virtual Camera from the Real Camera Parameters

This section describes how to model OpenGL's virtual perspective projection pinhole camera. While this section describes how to derive the OpenGL camera parameters from a calibrated camera, these methods can be adapted for any implementation of a virtual pinhole camera.

3D object-space coordinates (coordinates local to the object) undergo a series of transformations to window-space (screen-space) similar to the transformations described in Section 3.4. The ModelView matrix is a 4x4 matrix which specifies the transformation from object-space coordinates to eye-space (i.e. camera-space) coordinates:

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}. \tag{3.5}$$

Eye-space coordinates are transformed to clip-space coordinates (coordinates used to clip objects or portions of objects) through the camera projection matrix, which consists of four parameters. This projective transformation in matrix notation is as follows:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{\cot(fovy/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(fovy/2) & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2*zFar*zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}, \tag{3.6}$$

where *fovy* is the field of view angle in degrees in the y direction, *aspect* specifies the ratio of width to height, *zNear* is the distance from the viewer to the near clipping plane, and *zFar* is the distance from the viewer to the far clipping plane. Equation 3.6 can be alternatively written as:

$$\begin{aligned} x_c &= \frac{\cot(fovy/2)}{aspect} * x_e, \\ y_c &= \cot(fovy/2) * y_e, \\ z_c &= \frac{zFar+zNear}{zNear-ZFar} * z_e + \frac{2*zFar*zNear}{zNear-zFar}, \\ w_c &= -z_e. \end{aligned} \tag{3.7}$$

The resulting clip-space coordinates are divided by w_c (i.e. perspective division) to produce normalized device coordinates:

$$\begin{aligned} x_{nd} &= \frac{\cot(fovy/2)}{aspect} * \left(-\frac{x_e}{z_e}\right), \\ y_{nd} &= \cot(fovy/2) * \left(-\frac{y_e}{z_e}\right), \\ z_{nd} &= \frac{zFar+zNear}{zNear-ZFar} * (-1) + \frac{2*zFar*zNear}{zNear-zFar} * \left(-\frac{1}{z_e}\right), \\ w_{nd} &= 1. \end{aligned} \tag{3.8}$$

Normalized device coordinates are finally translated and scaled through the viewport transformation to produce window coordinates as follows:

$$\begin{aligned} x_i &= (x_{nd} + 1)\left(\frac{width}{2}\right) + x, \\ y_i &= (y_{nd} + 1)\left(\frac{height}{2}\right) + y, \end{aligned} \tag{3.9}$$

where x, y is the lower-left corner of the viewport rectangle in pixels, *width* and *height* are the width and height of the viewport, respectively, u_v, v_v are the calculated window coordinates. Substituting for x_{nd} and y_{nd} and expanding, u_v and v_v become:

$$\begin{aligned} x_i &= \frac{\cot(fovy/2)}{aspect} * \left(-\frac{x_e}{z_e}\right) * \frac{width}{2} + \frac{width}{2} + x, \\ y_i &= \cot(fovy/2) * \left(-\frac{y_e}{z_e}\right) * \frac{height}{2} + \frac{height}{2} + y. \end{aligned} \tag{3.10}$$

If the intrinsic and extrinsic parameters of the real camera are known, the virtual camera can be modeled such that a pixel (x_i, y_i) in I_v and I_r correspond to the same 3D point. Recall from Section 3.4, a pinhole camera projects a 3D world point to a pixel on the image plane through the following equation (assuming no skew or radial distortion):

$$\omega \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}. \tag{3.11}$$

Notice that the extrinsic parameters correspond directly with the OpenGL ModelView matrix and can be used for the eye-space transformation, leaving:

$$\omega \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} \tag{3.12}$$

which can be written as:

$$x_i = f * \frac{x_e}{z_e} + c_x, \quad y_i = f * \frac{y_e}{z_e} + c_y, \tag{3.13}$$

defining the link between the intrinsic parameters of the real camera with the virtual camera. However, there are a couple additional considerations before the virtual parameters can be solved for. First, the real camera is modeled with a viewing direction along the vector (0, 0, 1) while OpenGL's is along (0, 0, -1). A scaling matrix can be multiplied to the OpenGL's eye-space coordinates so that the two coordinate systems coincide:

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}. \tag{3.14}$$

x_i and y_i for the virtual camera now become:

$$\begin{aligned} x_i &= \frac{\cot(fovy/2)}{aspect} * \left(\frac{x_e}{z_e} \right) * \frac{width}{2} + \frac{width}{2} + c_x, \\ y_i &= \cot(fovy/2) * \left(-\frac{y_e}{z_e} \right) * \frac{height}{2} + \frac{height}{2} + c_y. \end{aligned} \tag{3.15}$$

There is also one more difference, illustrated by Figure 3-5.

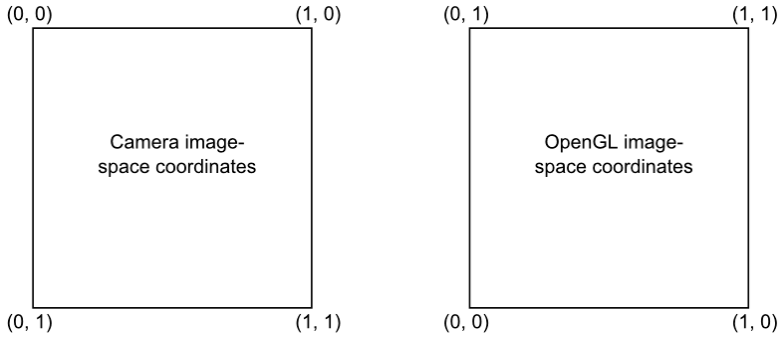


Figure 3-5: Image-space coordinate frames for the real and virtual cameras.

The real camera uses the upper left position of the image as its pixel origin while OpenGL uses the lower left position. Now comparing the two camera models:

$$f * \frac{x_e}{z_e} + c_x = \frac{x_e}{z_e} * \frac{\cot(fovy/2)}{aspect} * \frac{width}{2} + \frac{width}{2} + x, \tag{3.16}$$

and since $v_r = height - v_v$

$$f * \frac{y_e}{z_e} + c_y = height + \frac{y_e}{z_e} * \cot(fovy/2) * \frac{height}{2} - \frac{height}{2} - y. \tag{3.17}$$

From the above equations we see that:

$$\begin{aligned} c_x &= \frac{width}{2} + x, \\ c_y &= height - \frac{height}{2} - y, \\ f * \frac{x_e}{z_e} &= \frac{x_e}{z_e} * \frac{\cot(fovy/2)}{aspect} * \frac{width}{2}, \\ f * \frac{y_e}{z_e} &= \frac{y_e}{z_e} * \cot(fovy/2) * \frac{height}{2}. \end{aligned} \tag{3.18}$$

Solving for x, y, $fovy$, and $aspect$, the resulting OpenGL projection parameters are:

$$
\begin{aligned}
x &= c_x - \frac{width}{2}, \\
y &= \frac{height}{2} - c_y, \\
fovy &= 2 * \cot^{-1}\left(\frac{2 * f}{height}\right), \\
aspect &= \frac{width}{height}.
\end{aligned}
\tag{3.19}
$$

Now the intrinsic camera parameters (f, c_x, c_y) can be used to model the projection matrix of the virtual camera. This, along with defining the ModelView matrix from the camera pose, provides the means to align real and virtual images together. In the next chapter we discuss a texture mapping method to place camera images onto the extracted mesh from these aligned viewpoints.

Chapter 4 – Mesh Parameterization

4.1 Texture Mapping

Texture mapping is a technique used in computer graphics to add image detail to surfaces only at the expense of small computational cost. It requires that the object's surface is parameterized such that points in 3D object-space are mapped to 2D image-space. Once this mapping is obtained, textures will remain on the correct location of the surface regardless of the position and orientation it is being viewed from.

One method for parameterizing polygonal meshes is to assign each vertex a set of texture coordinates that correspond to 2D coordinates within a specific image. When the mesh is rendered, texture coordinates for points on the face of triangles are interpolated from the appropriate vertices by the graphics hardware. These coordinates are then used to lookup the color in the texture image and finally mapped onto the polygonal surface.

Given a registered I_r-I_v pair, the mesh surface visible in I_v can be texture mapped by projecting the camera image I_r onto it, as if by a slide projector. The key here is to determine which vertices in the mesh are visible to the camera, and then use their position in window-space to parameterize the surface. This remaining portion of this chapter focuses on determining the texture coordinates from registered I_r-I_v pairs and texture mapping the surface mesh.

4.2 Texture Coordinate Calculation

From the registered viewpoint, the 3D position of a pixel in I_r can be retrieved using the depth value at the corresponding pixel position in I_v. However, the majority of pixels

will not project back directly onto the vertex positions in the surface geometry, but rather onto a polygonal face. Because we are only interested in the vertices such a method is impractical and generally inefficient. Alternatively, we can determine the window coordinates for each vertex from the registered viewpoint and use these coordinates as texture coordinates based on visibility.

Following the transformation pipeline in Section 3.5, let $v_o = (x_o, y_o, z_o, 1.0)^T$ be the object-space coordinates of a vertex in the mesh. v_o is transformed into homogeneous clip-space coordinates $v_c = (x_c, y_c, z_c, w_c)^T$ through the following transformations:

$$v_c = P * M * v_o, \tag{4.1}$$

where M is a matrix that transforms object-space coordinates to eye-space coordinates and P is a projection matrix that transforms eye-space coordinates to clip-space coordinates.

From here, two visibility tests are performed to determine if the vertex contributes to the surface seen in I_r. The first test ensures the projected coordinates reside within the camera viewing frustum. Vertex clip coordinates within the camera's viewing frustum satisfy the following conditions:

$$-w_c \le (x_c, y_c, z_c) \le w_c. \tag{4.2}$$

For vertices within the viewing frustum, window coordinates (s, t, r) within the range [0, 1] calculated as follows:

$$s = 0.5 * \left(\frac{x_C}{w_C}\right) + 0.5, t = 0.5 * \left(\frac{y_C}{w_C}\right) + 0.5, r = 0.5 * \left(\frac{z_C}{w_C}\right) + 0.5, \tag{4.3}$$

where r is the depth along the camera's viewing axis, perpendicular to the st-plane.

The second test determines whether or not a vertex within the camera viewing frustum is occluded by a surface (e.g. by overlapping triangles). To check for occlusion, the isosurface is rendered into a depth buffer storing the nearest depth values. Each vertex value r is compared against the value d at position (s, t) in the depth buffer. r values equal to d lie on the observed surface while values greater than d are occluded.

Triangles which have all vertices within the viewing frustum and are not occluded by a surface are considered fully visible. The vertices of these triangles are assigned their (s, t) values as 2D texture coordinates, as illustrated by Figure 4-1. An additional index value i is given to these vertices to reference the correct texture image.

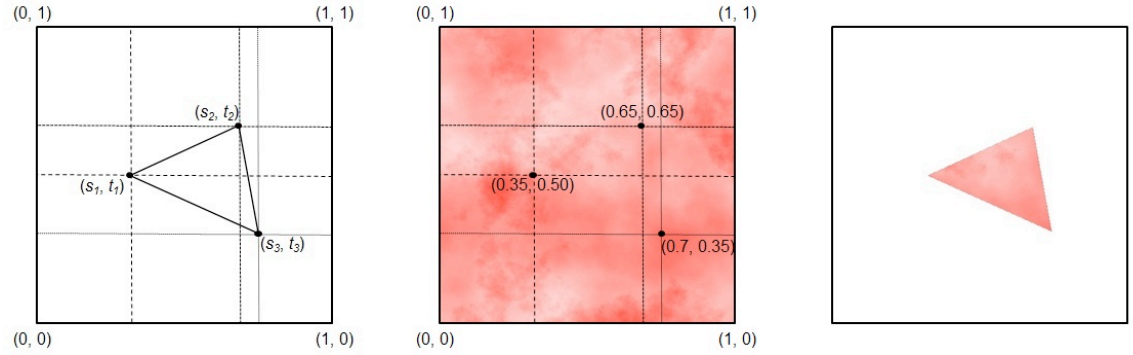


Figure 4-1: Projective texture mapping process. Given a registered I_r-I_v pair, each vertex in a triangle (left) is assigned a (s, t) pair of texture coordinates corresponding to the window position in the camera image (middle). The image detail is then mapped onto the triangle when rendered (right).

4.3 Texture Mapping Quality Assessment

It is important to evaluate the quality of the texture mappings when multiple images are projected onto a triangle. Naively texture mapping could replace a good result with one of poor quality. Factors such as the camera angle and distance with respect to the surface affect the amount of information that is texture mapped. In this case, the optimal

texture for a triangle is given when its projected area is maximized on the image plane, satisfying the following conditions:

$$\begin{aligned} cos^{-1}(n_s \cdot z_{view}) &= \pi, \\ \|p_s - p_{cam}\| &= z_{near}, \end{aligned} \tag{4.4}$$

where z_{view} is the direction of the camera's viewing axis, n_s is the surface normal, p_s and p_{cam} are the 3D world space position of the surface point and camera, respectively, and z_{near} is the distance to the near clipping plane. In other words, the best texture is given when the camera's viewing axis is perpendicular to the surface and is as close as possible to the surface, which would map the most image pixels onto the surface.

Distance and angle values can be stored for each triangle as they are texture mapped to evaluate the texture mapping quality of overlapping images. If the incoming image has better quality, it can be used to replace the old texture map result. Alternatively, Equation 4.4 can be packed into a single weight value

$$w_i = -\frac{(z_{far} - z)}{z_{far}} \cos(\theta), \tag{4.5}$$

where z_{far} is the distance to the far flipping plane, z is the depth of the surface point, and $\cos(\theta)$ is the dot product of the surface normal and camera viewing direction. The weight can be used not only for evaluating the quality of texture mapping, but also to perform blending between multiple images.

Angle and distance weighting can be additionally assigned based on the application. For example, camera distance may heavily influence the result when photographs are taken on the outside of a patient, since the camera position can vary significantly from one image to the next. On the other hand, endoscopic video tends to be the opposite since

the camera is generally always close to the surface. Therefore, the projection quality would be heavily influenced by the angle of the camera with respect to the surface, opposed to the distance.

4.4 GPU-Based Implementation

The texture mapping method described above consists of many independent calculations and frequent data transfer between the CPU and GPU to update texture coordinates. For applications where images are streamed in real-time, this overhead can reduce overall performance. Processing time can be reduced by exploiting the GPU to perform projection calculations in parallel and promote efficient data transfer by using fragment shader output to update values directly on the GPU.

In order to do this, two textures are created with a width and height of *ceil*(*sqrt*(*N*)), where *N* is the number of vertices in the surface mesh. One texture, called the vertex coordinate image, stores the object-space coordinates (x_o, y_o, z_o, 1.0) for each vertex into the RGBA components, respectively, and is passed to a fragment shader. The values stored in this image are static and do not change. The second texture, called the texture coordinate image, is used to store the texture coordinate values (*s*, *t*, *i*, 1.0) for each vertex and will be updated dynamically as images are texture mapped onto the mesh. This texture is set as a color render target to be written into by the output of the fragment shader. The RGBA values for the texture coordinate image are initialized to (-1.0, -1.0, -1.0, 0.0), where an A value of 0.0 indicates the vertex has not been texture mapped. This process is illustrated in Figure 4-2.

Processing is performed by reshaping the viewport to the width and height of the textures and rendering a full screen quad. This allows each fragment to perform projection calculations for one vertex and output the calculated texture coordinates in its RGBA components. Fragments processed where vertices are determined to be part of clipped, occluded, or partially visible triangles are discarded so that the values currently stored in the texture coordinate image are retained. As the mesh is texture mapped with camera images, the texels in the texture coordinate image are filled with new values for the appropriate vertices. A pixel buffer object is then used to update the texture coordinate values for mesh vertices from the texture coordinate image without involving CPU cycles. As a result the time necessary to perform these calculations is significantly reduced.

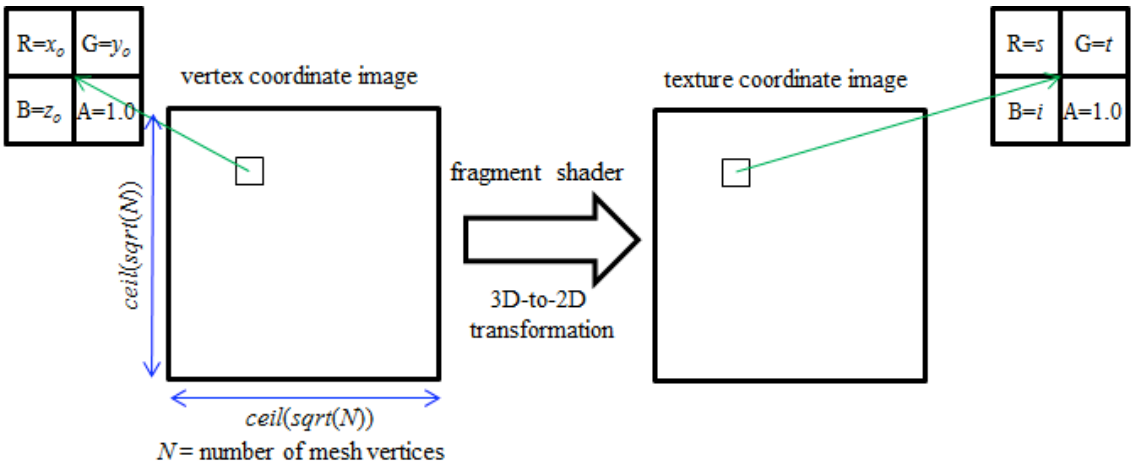


Figure 4-2: Vertex (left) and texture coordinate (right) images used to calculate texture coordinates on the GPU.

Chapter 5 – Surface Painting

The texture mapping method described so far is limited to storing only as many images as system memory will allow. This is not a practical solution for applications, such as endoscopic procedures, where multiple images must be stitched to the surface over a period of time. Consider a camera capturing 640 x 480 24-bit RGB images at 30 frames per second. This will require approximately 27.7 MB of memory to be allocated per second to store every image. Such a method is obviously impractical and inefficient as storing each image would require a considerable amount of memory and would be wasteful due to the redundancy of information between overlapping images.

A second drawback is the inability to properly texture map triangles that are not completely visible. This issue is shown by Figure 5-1. A triangle is considered partially visible when part of its surface is visible from a given viewpoint while the remaining portions are either occluded by a surface or clipped by the viewport. These triangles will consist of vertices with either undefined texture coordinates or texture coordinates belonging to more than one image, causing interpolation errors and an incorrect rendering. Because of this, it is necessary to exclude these triangles during texture mapping of the current projected image. However, this discards valuable information from the camera image and creates the appearance of holes or gaps that degrade the texture mapped result.

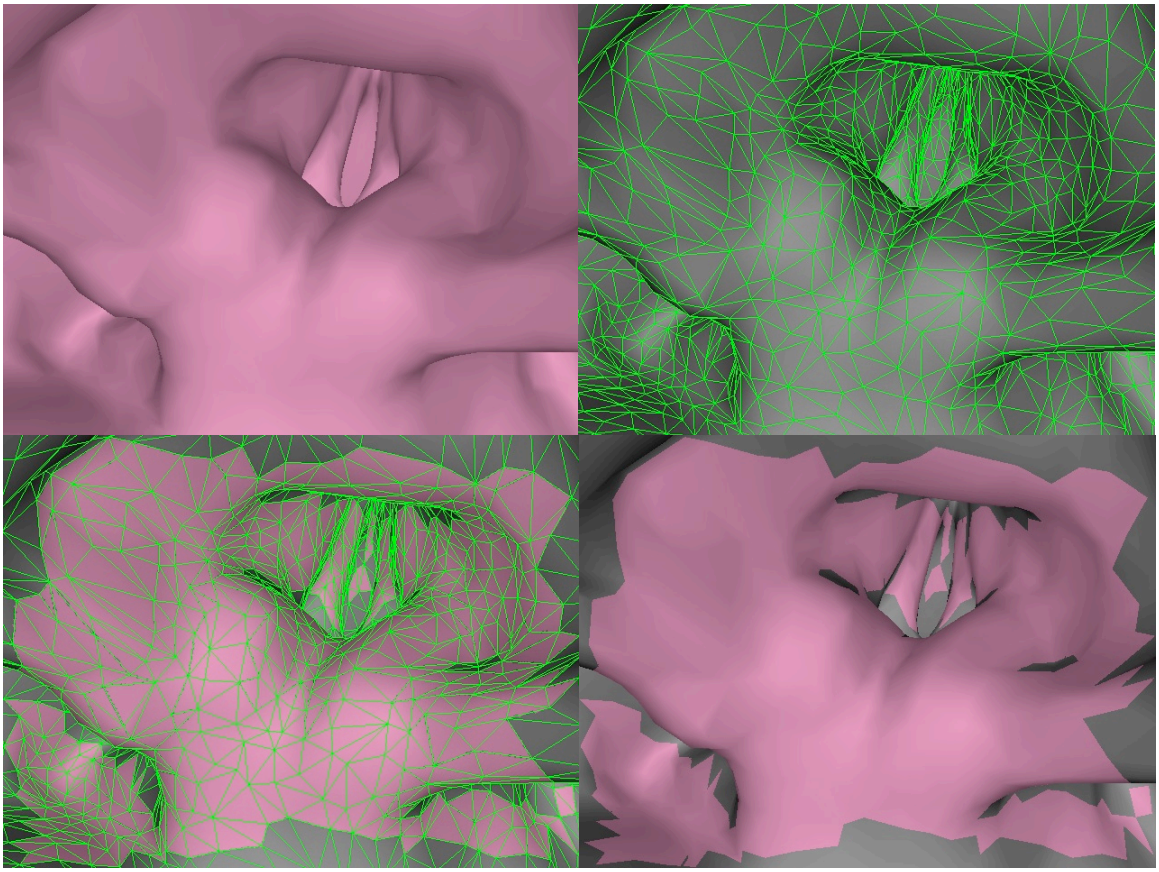


Figure 5-1: A camera image (top left) projected onto the surface mesh (top right) results in holes and gaps from removing partially visible triangles during projective texture mapping (bottom row). The lower left image shows the mesh wireframe overlaid on the result for visual inspection while the lower right shows the actual texture mapped mesh.

5.1 Texture Atlas

To overcome these limitations, we paint image color into a texture atlas. A texture atlas is a large image that contains smaller sub-images and is often used for texture mapping 3D models in applications such as games and computer animation. The same principle can be applied to the mesh model extracted from the volume provided that we obtain a parameterization for the mesh to the atlas, a process known as UV mapping. This will allow us to store images, as they are acquired, to the corresponding regions of the texture atlas rather than storing each individual image. Furthermore, we can take

advantage of this representation to paint portions of triangles into the atlas so the mesh can be rendered with triangles that are partially texture mapped.

There are several methods for generating UV-maps, which consist of segmenting the mesh into charts [73, 74, 75, 76, 77, 78], chart parameterization [78, 79], and packing the charts in texture space [78, 80, 81]. One of the most widely used is the least squares conformal maps (LSCM) algorithm [82], which is an automated method for generating a parameterization for a mesh to an atlas. This unfolds the 3D mesh and packs it into a 2D texture space, providing (u, v) parameters for each vertex to the atlas, while minimizing the angle deformation and non-uniform scaling of triangles. Figure 5-2 shows an example uv-layout generated for a surface mesh using the LSCM method. Since atlas generation cannot guarantee the area of the triangle in the atlas will match that in the mesh, the resolution of the atlas can be increased as necessary to ensure the desired level of image detail is retained.

Next, we describe the surface painting method used to write image color into the atlas at the fragment level.

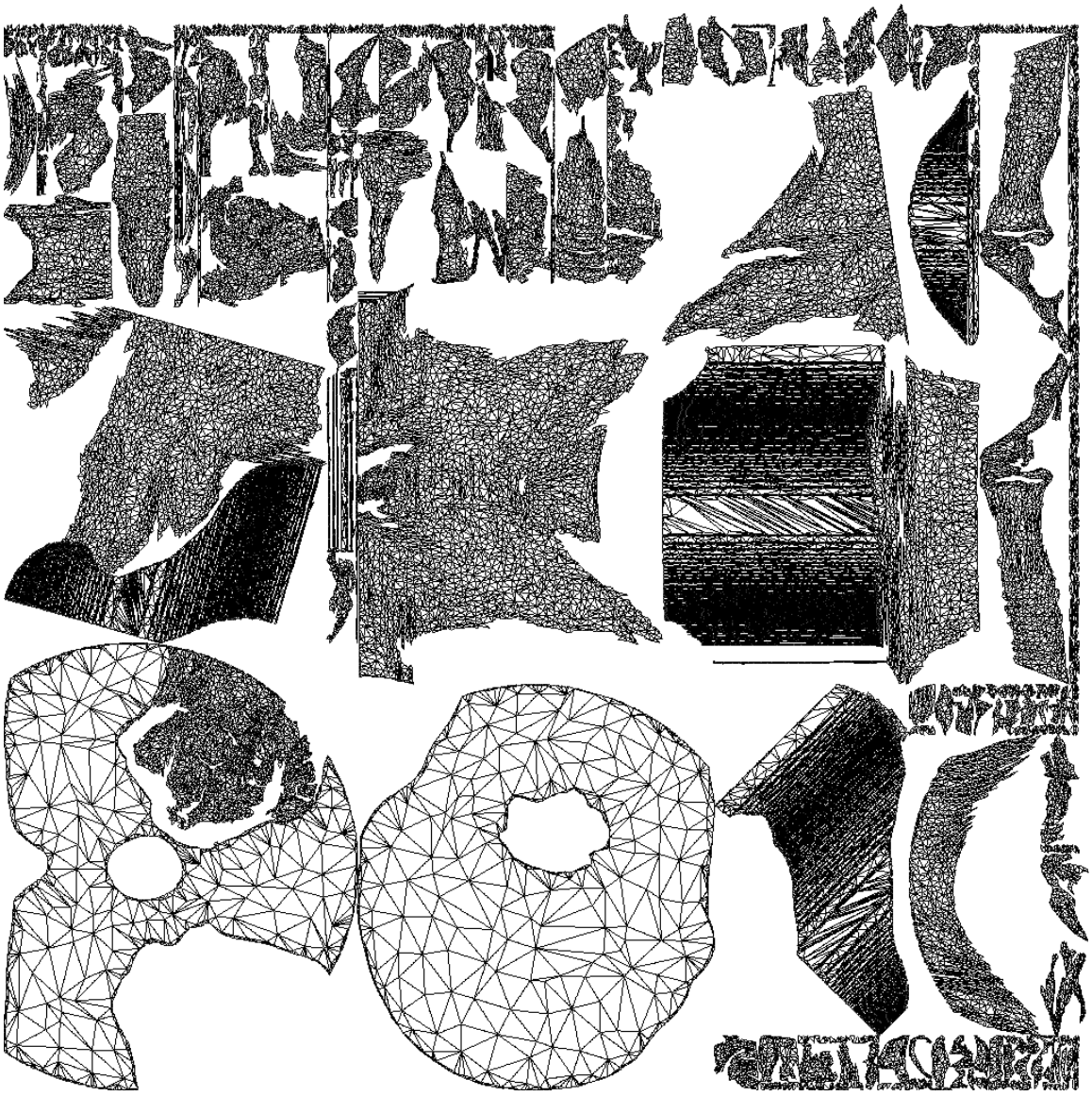


Figure 5-2: uv-layout for a surface mesh using least squares conformal maps.

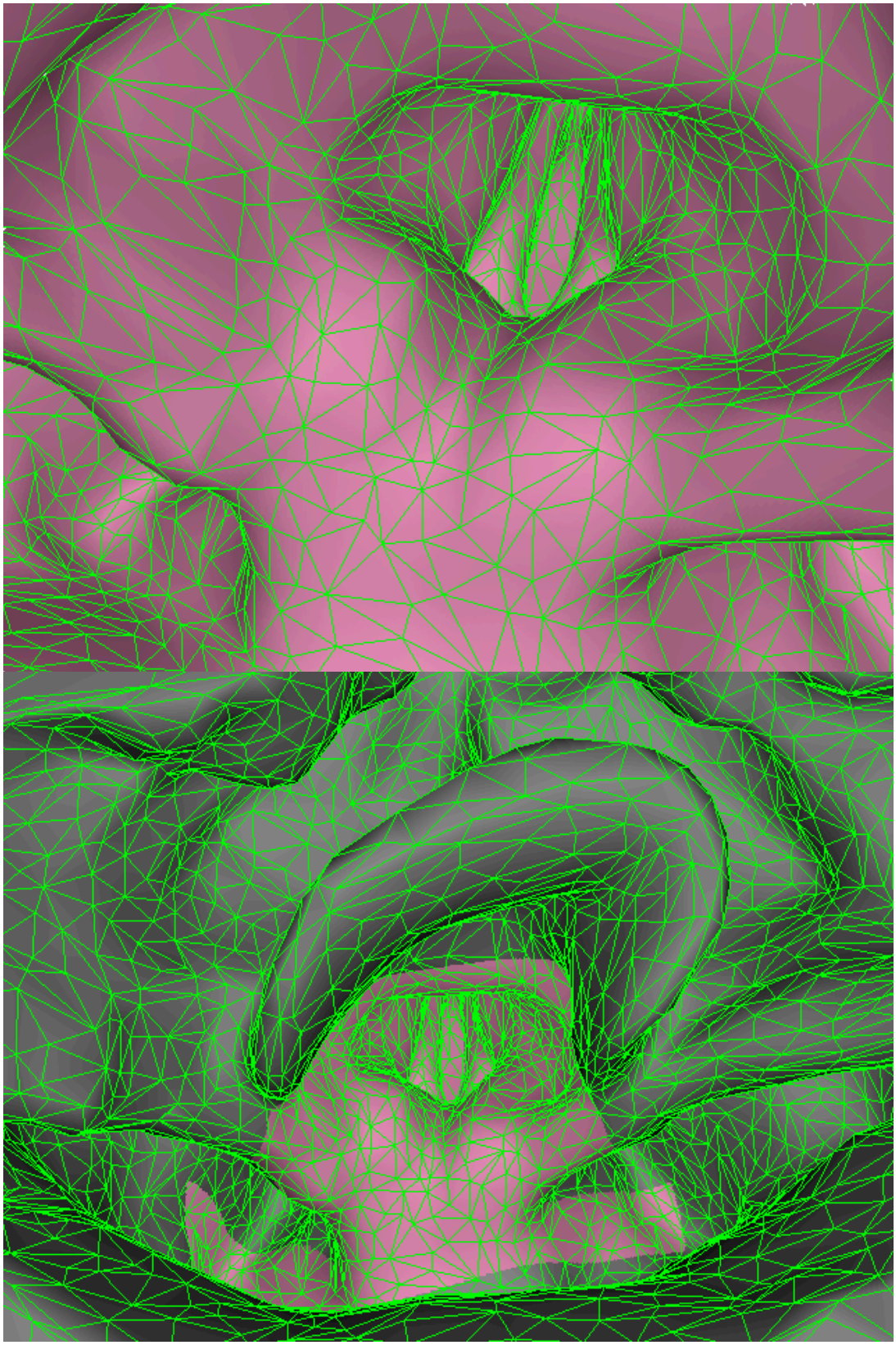
5.2 Surface Painting Algorithm

To implement painting into the atlas, we modify the texture mapping method described in Chapter 4 as follows:

1. Initialize the RGBA color of each pixel in the atlas to (0.0, 0.0, 0.0, 0.0).

2. Calculate (s, t, r) for each mesh vertex. For each triangle, if at least one of its vertices is inside the view frustum, assign the texture coordinates of each vertex in that triangle to its (s, t, r) value, appending a 1.0 as the 4^{th} component (the image reference i is not needed when using the atlas). Otherwise, if no vertices are visible assign each vertex (-1.0, -1.0, -1.0, 0.0).
3. Set the texture atlas as a color render target and reshape the viewport to the width and height of the atlas. Using a fragment shader, render the mesh using its (u, v) atlas coordinates as 2D vertex positions. Pass the current camera image and depth buffer to this fragment shader.
4. Check if (s, t) for a fragment are both within the range [0, 1] and if r is equal to the depth d at coordinate (s, t) in the depth buffer. If either of these tests fails discard the fragment.
5. Fetch the RGB value from the camera image at (s, t). This RGB value can replace, be blended with, or processed with any color values previously stored at the fragment level (Section 4.3). Output the final fragment color into the atlas using A=1.0.

Figure 5-3 shows the resulting effect of this algorithm when the mesh is rendered and the atlas is read from. The top image shows the partially visible triangles (from Figure 5-1) have been texture mapped with the image. The middle image shows a new viewpoint where visual inspection reveals that these polygons are partially painted. Furthermore, we can write any number of images into the atlas. The next step is to merge the color detail texture mapped onto the mesh with the original 3D anatomical data set to produce a volumetric rendering.



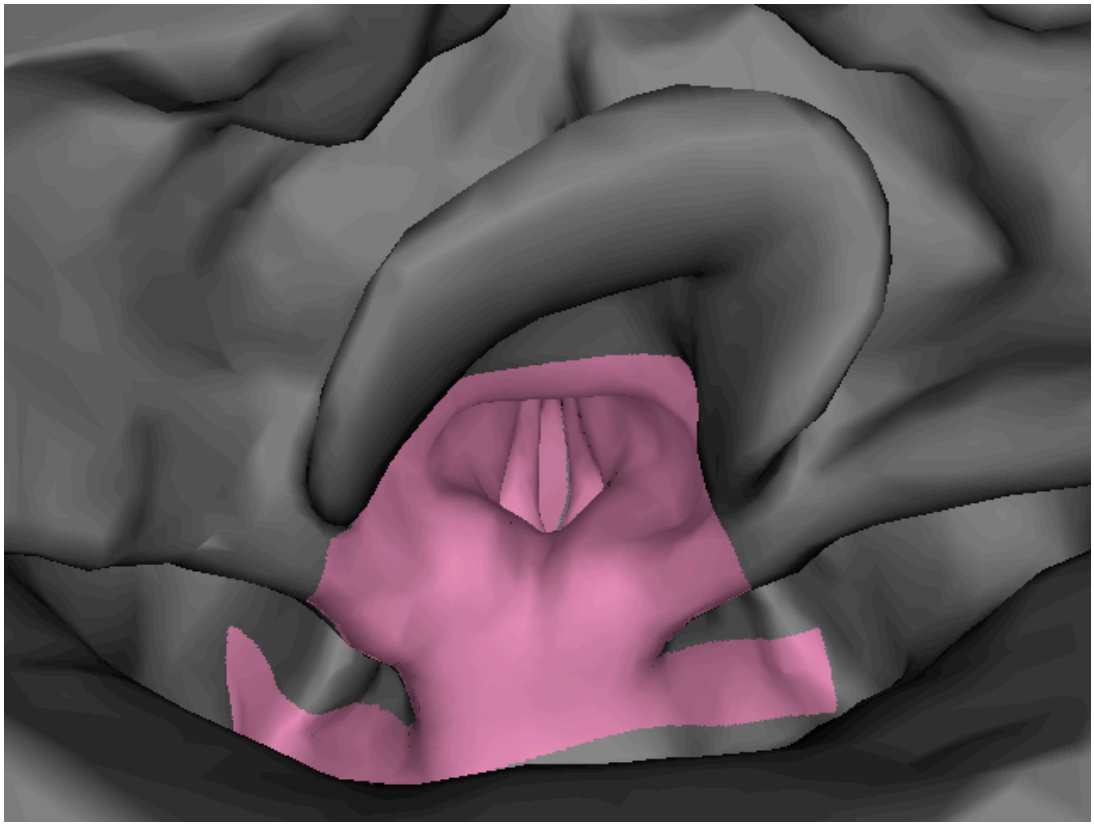


Figure 5-3: Texture mapping partially visible triangles using the surface painting method (top). The effect of this method can also be seen viewing the mesh from an alternate viewpoint, with wireframe overlay (middle) and without (bottom).

Chapter 6 – Volume Rendering using GPU-Based Polygon-Assisted Raycasting

6.1 Volume Rendering

In principle, the idea behind volume rendering is to evaluate the volume rendering integral

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(t) e^{-\int_{s}^{D} \kappa(t)dt} ds, \tag{6.1}$$

which describes the interaction between light and the participating media. The term I_0 represents the light entering the volume from the background at position $s = s_0$. $I(D)$ is the radiance leaving the volume at s = D and reaching the camera. The first term calculates the light from the background attenuated by the volume. The second term is the integral contribution of the source terms attenuated by the participating medium along the remaining distances to the camera [83].

In practice, volume rendering techniques aim to provide an approximate evaluation of this integral, since it generally cannot be determined analytically. Compositing is one method used for iterative computation of the discretized volume rendering integral. In this chapter, we focus on the volume rendering technique known as raycasting in which view rays are traversed from the eye point into the volume. Because of this, compositing is performed in a front-to-back order. The front-to-back iterative compositing equations are

$$\begin{aligned} C_{dst} &= C_{dst} + (1 - \alpha_{dst}) * \alpha_{src} * C_{src}, \\ \alpha_{dst} &= \alpha_{dst} + (1 - \alpha_{dst}) * \alpha_{src}, \end{aligned} \tag{6.2}$$

where C_{src}, α_{src} are the color and opacity of classified volume samples and C_{dst}, α_{dst} are the accumulated color and opacity, respectively.

In the case of 3D medical data sets, the vast majority of voxel data is a single scalar value representing some spatially varying physical property with unknown optical properties. Since there is no method for obtaining these properties, the user decides the appearance of structures by assigning optical properties to the scalar values using arbitrary mappings, known as a transfer function. The most general transfer function will map scalar values to a four-channel RGBA (red, green, blue, alpha) color value required for the above compositing equations. Multidimensional transfer functions can also be used. For example, additionally using the first derivative (i.e. gradient) and second derivative (i.e. curvature) of voxel values to perform these mappings.

6.2 GPU-Based Polygon-Assisted Raycasting

In the previous chapters, we discussed how to interactively project textures onto a surface mesh obtained from the volume. However, since our goal is to create volume rendered views of the patient, the color texture mapped onto this mesh must be combined with the values in the original volume data set. For the purposes of interactive visualization, it is also vital for the augmented volume to be rendered in real-time. In order to do this, we leverage modern graphics hardware as well as utilize the polygonal mesh to accelerate raycasting in addition to adding color detail to the volume. Because of this we refer to our technique as GPU-based polygon-assisted raycasting.

Figure 6-1 illustrates the GPU-based polygon-assisted raycasting method. This approach is an extended variant of the polygon-assisted raycasting algorithm [54] and the

GPU version proposed by Leung et al. [55] which use mesh geometry to reduce empty-space samples. For each pixel, a single ray is cast into the volume. Volume sampling begins on the mesh at positions such as D and G and ends at positions such as E and J, which tightly fit to potential visible regions of the volume. As the volume is sampled between these locations, the color (if any) texture mapped on the mesh is composited with the classified volume samples.

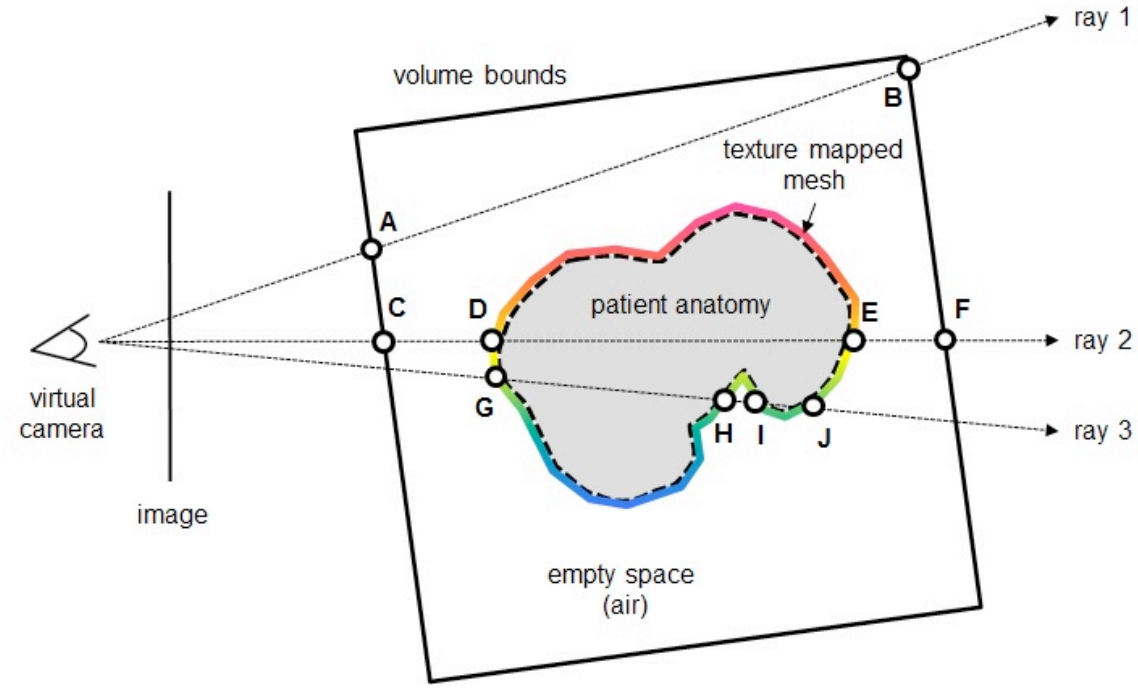


Figure 6-1: GPU-based polygon-assisted raycasting. Colored regions represent texture mapped mesh. Ray traversal is performed between ray segments such as DE, GH, and IJ, compositing the mesh color with volume samples. Ray segments which sample empty-space, such AB, CD, and EF, and HI are completely avoided.

This process consists of two stages: First, the mesh is rendered from a user-specified viewpoint, storing depth and color layers. The depth values retrieved from this stage provide the means to set up each viewing ray, including the ray start-end positions and the ray direction. Second, a GPU-based raycasting pass to sample the volume and

combine the texture mapped mesh colors in the correct visibility order. These stages are described in the two following subsections.

6.2.1 Ray Setup

Before raycasting can be performed, each viewing ray must be set up according to the camera parameters and respective pixel position. We must also combine the mesh color with the volume. Using the nearest front faces and furthest back faces of the mesh alone will not adequately capture all mesh colors along the viewing ray. For example in Figure 6-1, only colors at ray entry and exit points such as G and J can be stored, but any colors at intersections in between these, such as H and I, are missed, leading to an incomplete compositing result. Instead, the color and position of each ray-mesh intersection must be captured so that the color can be composited in the correct front-to-back order with volume samples.

To acquire this information we employ a stencil routed *k*-buffer [84], which is a buffer used for achieving order independent transparency. A *k*-buffer is capable of capturing up to *k* fragments per pixel per rendering pass, where *k* is dependent on the graphics hardware. The *k*-buffer is implemented with a multisample texture and antialiasing disabled, and then using a stencil buffer to route incoming fragments to specific sample locations, providing a vector of fragments per pixel. One multisample texture is used to capture the color that was texture mapped onto the mesh and a second multisample texture is used to capture the depth values. Figure 6-2 shows an example of the first 4 nearest depth and color layers for a mesh from a given viewpoint using stencil routed *k*-buffers. Depth peeling [85, 86] is an alternative method to obtain depth layers,

where each layer is stored in a separate buffer, and will yield similar results. The difference between depth peeling and the stencil routed *k*-buffer is that it requires *n* mesh rendering passes (or *n* / 2 in the case of dual depth peeling), where *n* is the depth complexity of the object from a given viewpoint.

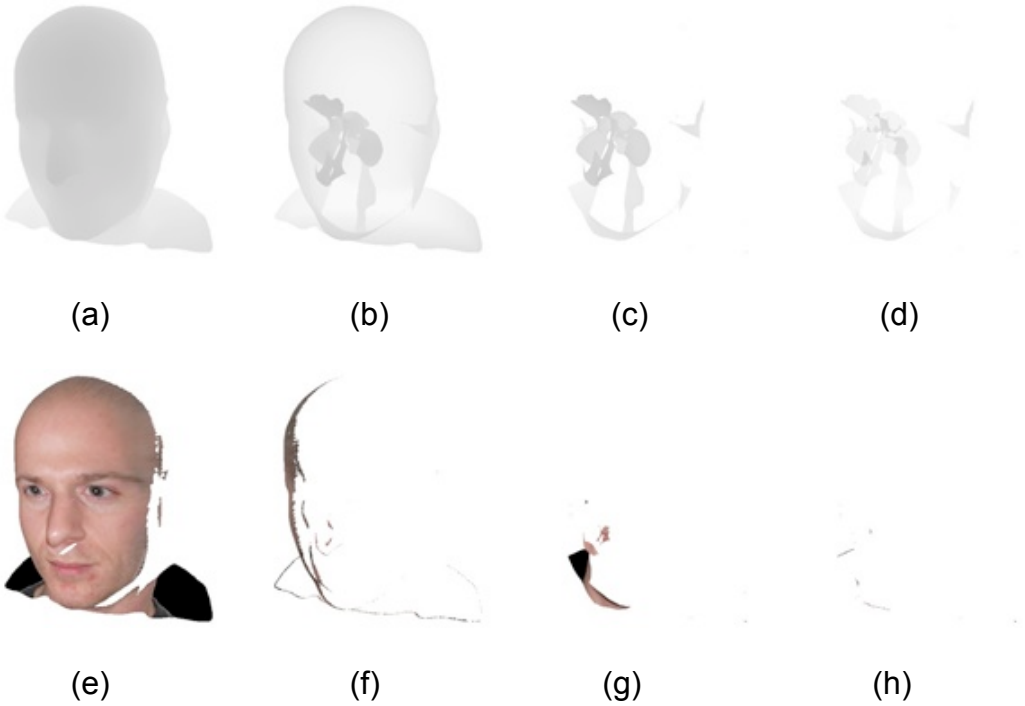


Figure 6-2: The nearest 4 depth (a-d) and color (e-h) layers extracted from a texture mapped mesh from a given viewpoint.

When the mesh is rendered, each vertex uses its (*u*, *v*) texture coordinates to read from the corresponding region in the atlas. Areas on the surface that have not been texture mapped are given RGBA values of (0.0, 0.0, 0.0, 0.0), which prevent color and opacity contribution during volume rendering. Surfaces that texture mapped are assigned the RGB color values stored in the atlas at the appropriate location and assigned an alpha value based on a user-specified transparency parameter α_{user}, which allows the user to control the level of camera image transparency in order to see through the surface. This

enables an effect similar to the visualization produced by Scharsach et al. [52] to allow see-through of isosurfaces in order to reveal interior structures. Once the multisample textures are filled with values, they are passed to a fragment program for a follow-on raycasting pass.

6.2.2 Raycasting

Inside the raycasting fragment program, the multisample color and depth texture values are sorted from nearest to farthest for each pixel. Depth values are back-projected with their pixel position to obtain 3D positions within the volume's local coordinate system. These positions correspond to each ray-mesh intersection. The nearest position is subtracted from the farthest position and normalized to obtain the ray direction.

The start and end positions for each ray segment are slightly offset along the ray direction to ensure relevant volume samples are not lost due the surface mesh approximation. As rays are traversed the volume is sampled at discrete step sizes. When the ray intersects the mesh C_{src} and α_{src} are set to the corresponding RGBA value in the color multisample texture. This allows the mesh color to be composited with the volume samples in the correct visibility order. Furthermore, this process allows us to merge camera image detail with the original volume independent of the volume's resolution.

6.3 Acceleration Techniques

Two factors that affect raycasting performance are the application resolution (fill rate) and number of samples taken for each ray. Since the applications resolution is determined by the user, we focus on improving the second factor. There are a number of acceleration

techniques that can be incorporated into raycasting algorithms. Two acceleration techniques commonly employed are empty-space skipping and early ray termination [48, 87, 88, 89], which focus on reducing the number of ray samples which do not contribute to the final image. Both of which can reduce memory access, save computational power, and significantly increase rendering performance.

6.3.1 Empty-Space Skipping

Volume data sets often contain a larger number of voxels that do not contribute to the final rendered image. The contents of these voxels could be air that was scanned with the patient or data values assigned alpha values of zero in the transfer function. Recall that the extracted surface mesh fits to potentially visible regions of the volume. Referring back to Figure 6, this avoids sampling between points such as C and D and E and F that would occur with GPU-based raycasting. Furthermore, rays which sample empty-space outside the mesh geometry, such as ray 1, are completely avoided. As a result, GPU-based polygon-assisted raycasting can significantly reduce the number of empty-space samples.

6.3.2 Early Ray Termination

Early ray termination allows us to end ray traversal as soon as it is known that additional sampling will no longer contribute to the compositing result. Ray traversal for a pixel is terminated when the accumulated opacity reaches a user-defined threshold, typically when $\alpha_{dst} \geq 1.0$ or very close to 1.0. This is due to the fact that any additional sampling will no longer affect the final pixel color. This stopping mechanism is used in

addition to the termination test for when the ray exits the farthest depth layer of the surface mesh.

Chapter 7 – Results

This chapter presents experimental results using the proposed methods. The implementation was based on C++, OpenGL, and the Cg shading language. The application was run on a Dell Precision 690 Workstation with a Dual-Core Intel Xeon 5160 3.0GHz processor, 4 GB ram, NVIDIA Quadro FX 4600 graphics card.

The resolutions of the surface meshes used for the results below were determined by extracting a high-resolution geometry from the CT scans, and decimating them to a low enough triangle count where they still adequately bounded the volume. The sections below demonstrate the feasibility of the proposed methods for medical applications using camera images and 3D anatomical data sets.

7.1 Maxillofacial Surgery

The MANIX CTA data set (courtesy of OsiriX) was used to demonstrate the generation of 3D facial models. The 3D CT has size 512 x 512 x 343 with a pixel spacing of 0.48828125mm, 0.48828125mm, and 0.7mm, respectively. Some of the slices in this data set were omitted to reduce the overall size. The scalar values were scaled from 0 to 4095 and an isovalue of 600 was selected to extract the mesh. The final mesh used for this volume consisted of 63,716 triangles. Color photographs were simulated using surface renderings from the mesh. Color was added by encoding the object-space positions of each vertex into RGB values. Because the photographs were obtained from a virtual camera with known parameters viewing the same surface mesh from the CT, a perfect registration was obtained.

In Figure 7-1, three simulated photographs were texture mapped onto the volume using the surface painting method outlined in Chapter 5, and the weighted blending method in Section 4.3. (a-c) shows the painting process for left, right, and frontal portraits, respectively. The color image is shown on the bottom right, the registered virtual view is on the bottom left, and the result of the atlas after texture mapping the color image is shown on top for each. (d) shows a volume rendered view texture mapped with all three images and (e) shows the result after changing the image transparency and volume transfer function.

(a) (b) (c)

(d) (e)

Figure 7-1: Surface painting and blending using a texture atlas. (a-c) shows the painting process for left, right, and frontal portraits, respectively. The color image is shown on the bottom right, the registered virtual view is on the bottom left, and the result of the atlas after texture mapping the color image is shown on top for each. The result of the 3 images merged is shown in (d) with a user-defined image transparency and transfer function in (e).

Figure 7-2 demonstrates an example of a single real photograph texture mapped with a volume rendering of a CT angiogram used in the above example. The color and texture information in the photographs adds visual realism to the CT. Once one or more photographs have been textured onto the volume, the user can view the combined data volumetrically from arbitrary viewpoints, and change the appearance properties to create a visualization of their choosing. High-quality volume rendered views are generated in real time where the internal anatomy, such as bone, muscle, fat, and vascular structures can be clearly visualized in relation to the color images.

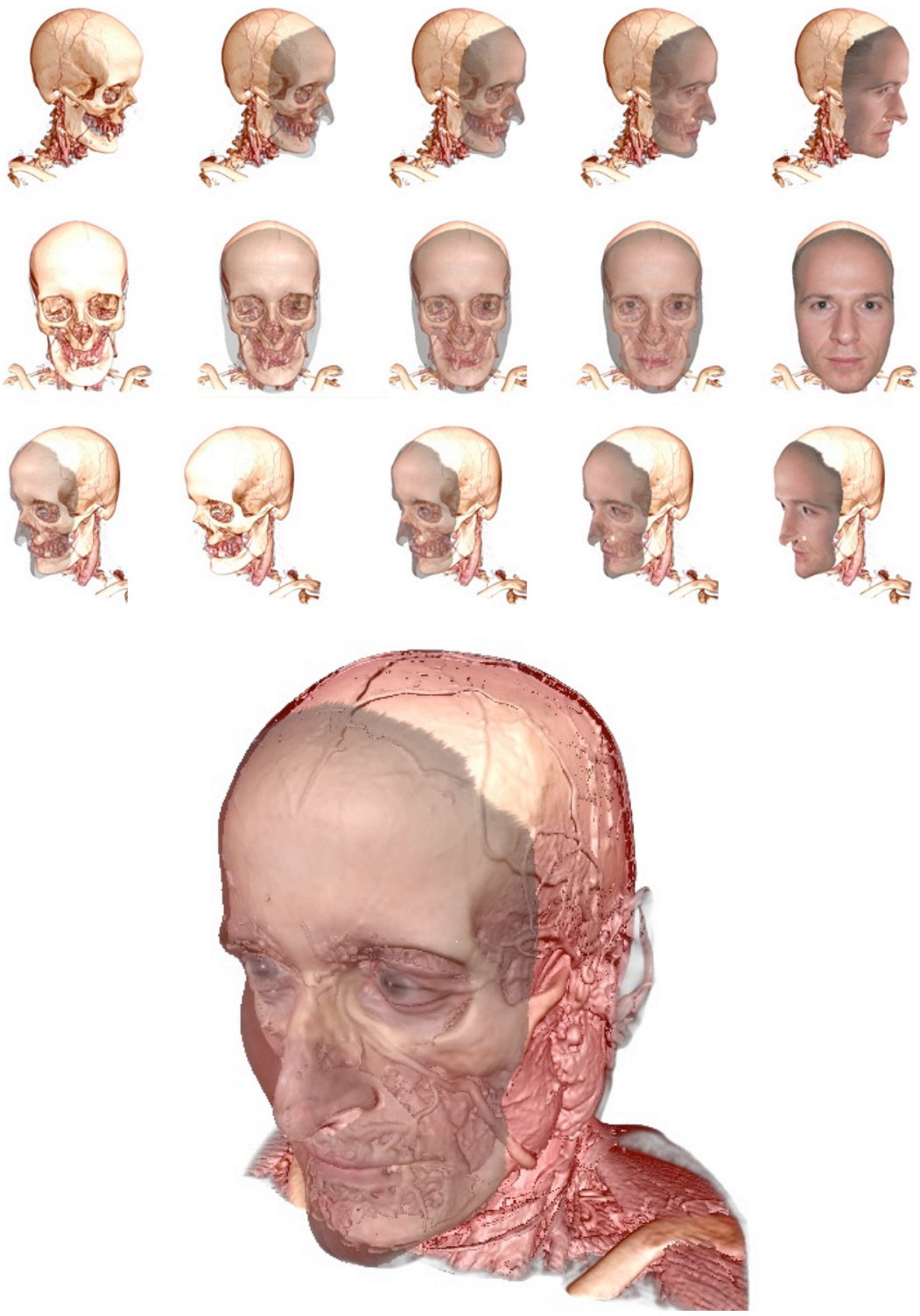


Figure 7-2: Single view photograph (top left) fused with a 3D CT angiograph (top right). Once the photograph has been texture mapped onto the volume it can be viewed from multiple viewpoints with varying levels of image transparency (middle 3 rows). High-quality volume rendered views are generated in real-time to show the spatial relationship of information (bottom).

7.2 Endoscopic Surgery

The preoperative data set used for this experiment is a 3D CT of a patient's neck with a size of 512 x 512 x 180 and spacing of 0.30273399mm, 0.30273399 mm, and 0.5999908mm respectively. The data values were scaled from 0 to 2874, and an isovalue of 600 was selected to extract the skin-air boundary from the CT. This resulted in a surface mesh of 194,726 triangles. Using this mesh, a rapid prototype of the neck was constructed out of ABS plastic using a Dimension SST 3D printer (http://www.dimensionprinting.com). Figure 7-3 shows the rapid prototype and the mesh.

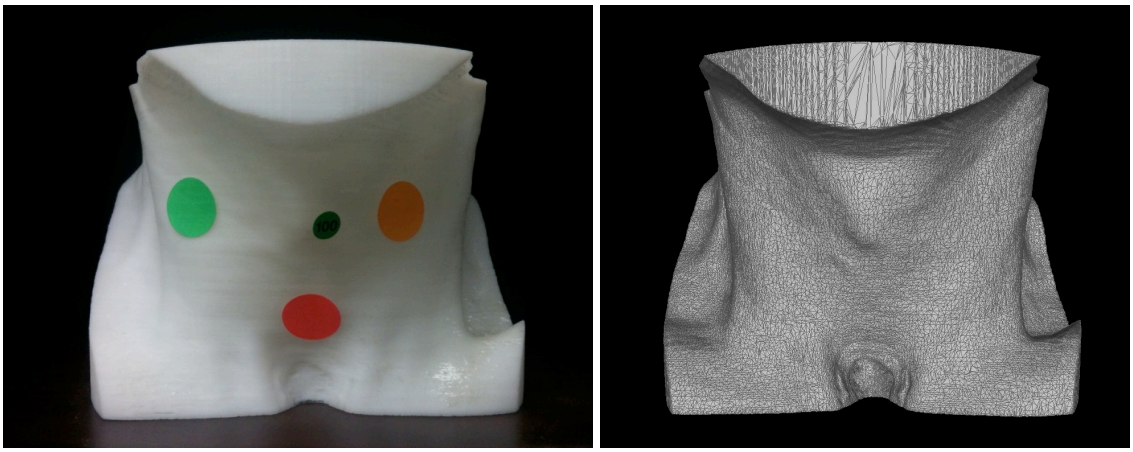


Figure 7-3: Phantom model (left) constructed from a surface mesh (right) extracted from a patient CT scan.

An Olympus ENF-P4 Rhino-Laryngo Fiberscope coupled to a Stryker 988 Camera System and a Matrox Vio Duo frame grabber card was used for capturing video. The scope was calibrated using a Tsai-based method [72] to obtain the camera's intrinsic parameters and model the virtual camera. The scope was inserted into the phantom model and video of the airway was recorded. Five frames from the video were selected as candidate images. From these images, five registered views of the surface mesh were

generated using the method proposed by Yim et al. [6, 7], using gradient-based mutual information for viewpoint matching between the endoscopic video frames and CT surface. The position and orientation of the endoscope was determined through this registration process. Figure 7-4 shows the registration results using this method.

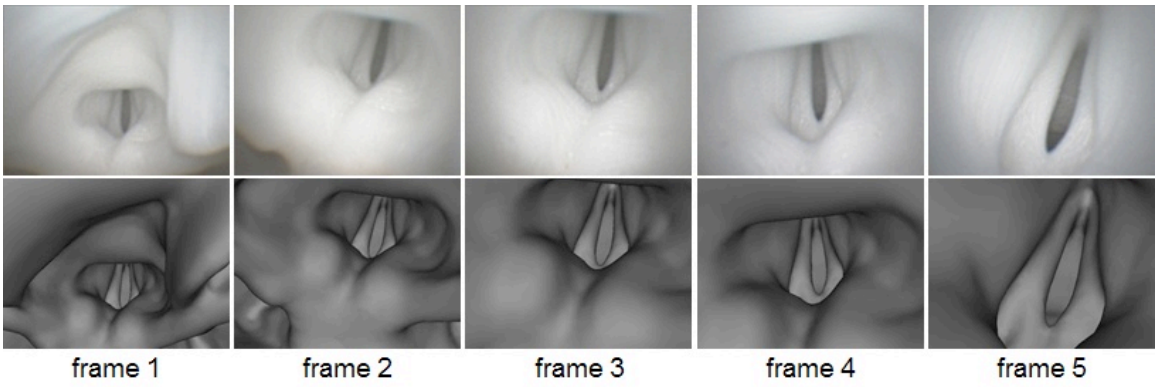


Figure 7-4: Sequence of five endoscopic video frames (top) and registered 3D CT surface renderings (bottom).

Texture mapping was performed by projecting each captured video frame onto the corresponding surface. The top image of Figure 7-5 shows the results after these images were stitched together, blended, and mapped onto the volume from a slightly zoomed out viewpoint. The bottom image shows a novel volume rendered view of the same fused data on the outside of the neck, where the video can be seen texture mapped on the airway.

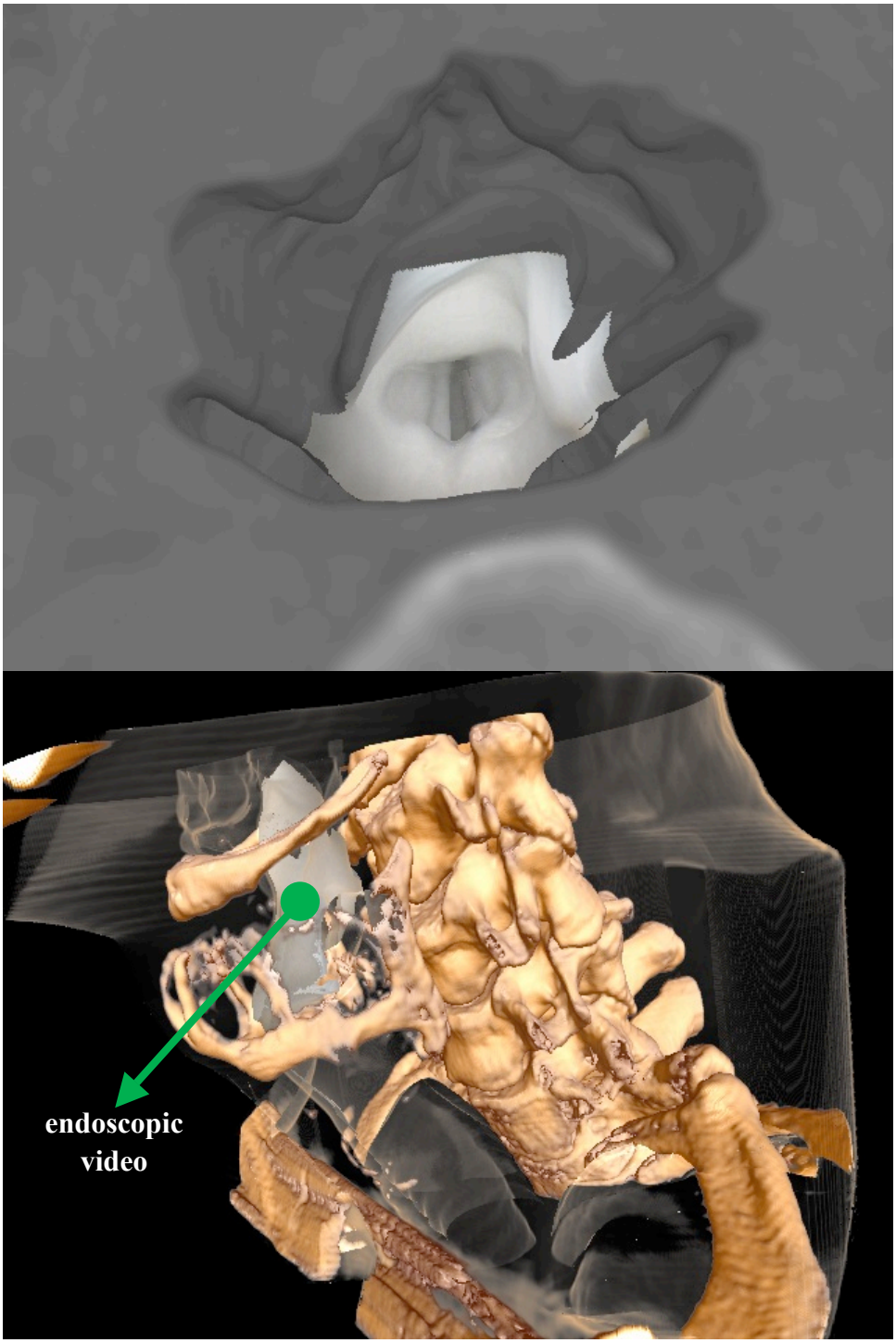


Figure 7-5: Volume renderings of registered 3D CT and endoscopic video images.

7.3 Performance

Performance was evaluated for the proposed GPU-based polygon-assisted raycasting algorithm and the surface painting methods to determine their efficacy for medical applications.

The GPU-based polygon-assisted raycasting technique was compared against standard GPU-based raycasting as the benchmark. Both volume rendering techniques were applied to the MANIX CT data set shown in Section 7.1. For both methods, the volume was sampled in 3D texture space with a step size of 0.002 and rendered using a screen resolution of 640 x 480. Frames per second were calculated by averaging the number of frames rendered over several seconds while rendering the volume from varying viewpoints. The volume was rendered using two transfer functions for each raycasting method, one to display the skin and one to display the bone in the CT. The results of the performance are given in Figure 7-6.

The results show that the GPU-based PARC method performed approximately 3.7 times faster than GPU-based raycasting for rendering skin, and approximately 2.8 times faster for rendering bone. GPU-based PARC performed fast for the skin because each ray begun in close proximity to non-empty voxels. This is because the bounding mesh was obtained from the skin surface and only a few samples were needed before the opacity threshold was reached. For bone, both methods performed slower due to the increase in empty voxels from the removal of skin and other soft tissue. The speed-up for GPU-based PARC was lower for bone than for the skin because the surface mesh does not fit tightly with the bone structures, resulting in additional empty-space sampling. However,

the method still easily outperformed GPU-based raycasting with the added benefit of adding textures to the volume.

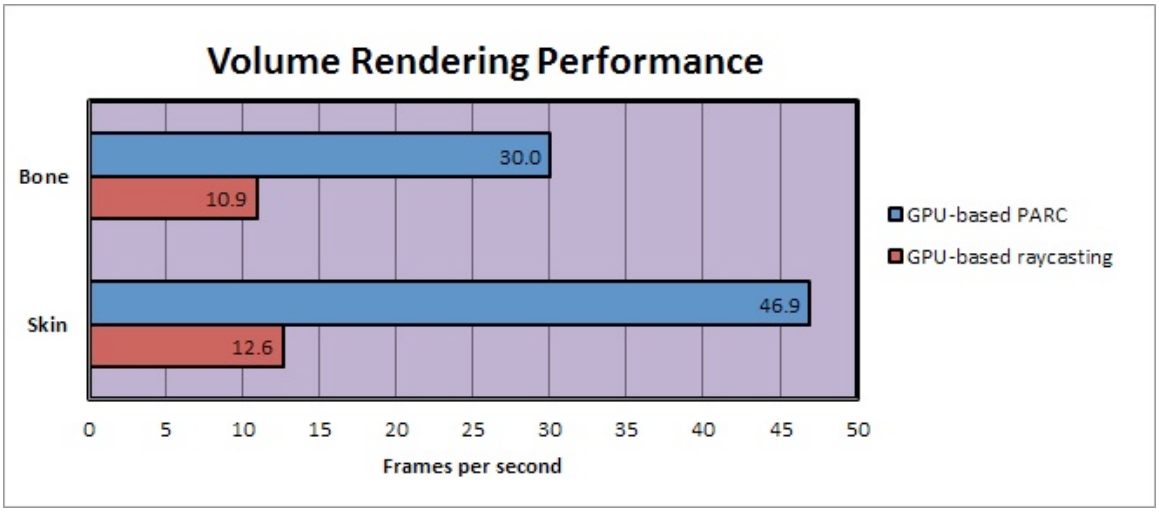


Figure 7-6: Performance comparison of the proposed GPU-based polygon-assisted raycasting method versus GPU-based raycasting using the MANIX CT data set. Bone and skin transfer functions were used to show the effects of both algorithms as the amount of empty space increases.

In the case of image-guided surgery, real-time registration and visualization is critical. The time taken to perform the proposed texture mapping and surface painting methods was calculated to test for viability in these types of systems that register images in real-time. Because the performance of these methods is dependent on the number of processed mesh vertices, the mesh resolution was varied in 25,000 triangles intervals, from 25,000 to 200,000, to see the effect for different mesh resolutions. The average times to texture map each mesh over 100 consecutive frames for a CPU and GPU implementation was calculated. These times included the time it took to paint each image into the atlas and blend it with previously mapped images.

In Figure 7-7, we see that there is a relatively linear increase in processing time as the resolution of the mesh increases. While the CPU implementation produced reasonable

texture mapping times, the GPU implementation provided a significant speed up. Furthermore, the mapping times for the GPU implementation only had a marginal increase in processing time as the mesh resolution increased when compared to the CPU version. These results show that camera images can be continually streamed onto the mesh, while it is being visualized, for small additional computational cost and is feasible for real-time systems.

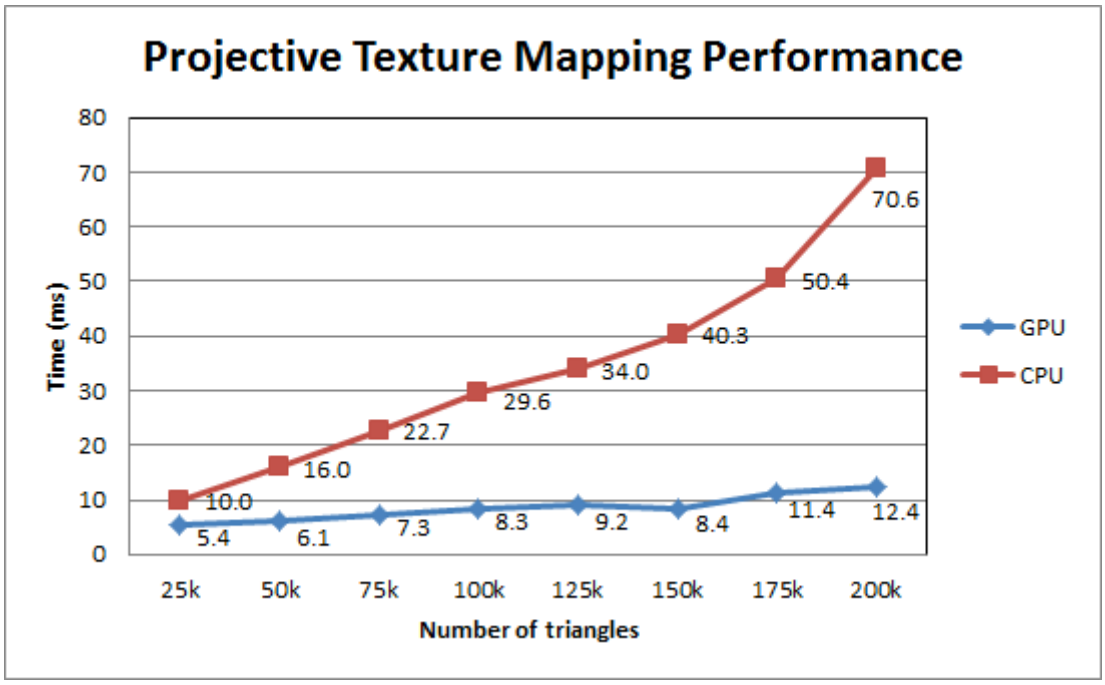


Figure 7-7: Projective texture mapping performance of CPU and GPU-based implementations for meshes of varying resolutions.

Chapter 8 – Conclusion and Future Work

8.1 Conclusion

This dissertation presents methods to create high-quality volume rendered views from registered 2D camera images and 3D anatomical data sets that enable realistic 3D visualization, navigation, and localization of information. The volume renderings contain the color and texture detail of the camera images with the structural information in the anatomical data set. By using a hybrid mesh-volume model, we were able to achieve interactive performance for rendering and handle the texture mapping of multiple images all in real-time. Furthermore, each generated patient model can be stored as a file to be assessed and analyzed at a later time. Due to the flexibility of the method from an implementation and application standpoint, the results are promising. It should be emphasized that the processing power of CPU's and GPU's is increasing quickly, thus these techniques can be significantly faster than the results shown here for the workstation.

The presented methods have a wide range of application for surgical planning, simulation, and image-guided surgery, but can also serve for valuable educational and training purposes. Because the tailored graphics techniques are similar to those frequently used for medical visualization, it is relatively straightforward to integrate them into systems that utilize this form of multimodal image fusion. Also, because volume visualization is enabled, this method is also ideal for medical applications that use additional 3D imaging modalities, such as positron emission tomography (PET). It is

envisioned that the use of these techniques will assist in improving treatment outcomes for a variety of medical procedures.

8.2 Future Work

The techniques described in this work provide solutions to some of the fundamental problems encountered in order to merge two-dimensional images with three-dimensional data sets. In addition to these methods, there are several potential extensions to explore.

Alternative visualization schemes, such as the focus + context, have been considered to display camera images in relation to the volume instead of using transparency alone. Since the camera image color is composited directly with volume samples, key features can become deemphasized at high transparency levels. Identifying methods that can emphasize these features and suppress less important ones, while retaining their context, could improve visualization and understanding of the information. An example of this would be the context preserving hotspot visualization technique proposed by Krüger et al. [90]

Photo mosaics are images generated from stitching a series of smaller images together so that a larger field of view of a scene can be obtained. The process of texture mapping several images onto a 3D surface is an analog of this. A technical challenge of generating a good mosaic is joining the images so that there is no visible edge or seam between them. Seams can be caused by vignetting, changes in surface illumination, registration errors, among others. In Section 4.3, some methods for evaluating texture quality were presented, however it would be worthwhile to explore alternative blending techniques to remove seams and obtain visually accurate results. The multiresolution spline technique

[91] is widely used method for creating visually pleasing blending results in image mosaic applications. Such a method could be applied to the stitched camera images of the patient. However, an important clinical consideration is to maintain relevant information so that the final result is not compromised. In this case, a more in depth study of color determination through the camera optics and interaction of light with the surface is required.

Finally, the major bottleneck of this approach is that performance is directly affected by the resolution of the surface mesh and number of layers that must be extracted to place color onto the volume. Current computer hardware is fast enough to render this information together for reasonably sized volume data sets at interactive rates. However, as the data increases in size and complexity, the feasibility of using such an approach decreases. It is important to note here that this is generally true with visualizing very large three-dimensional data sets. Extensions of the current approach include reducing the complexity of mesh and depth in irrelevant regions, which could provide a significant boost to performance and improve the robustness of the presented methods.

References

[1] C. B. Gossett, C. B. Preston, R. Dunford, and J. Lampasso, "Prediction accuracy of computer-assisted surgical visual treatment objectives as compared with conventional visual treatment objectives," *Journal of Oral and Maxillofacial Surgery*, **63**(5), pp. 609-617, 2005.

[2] R. T. Azuma, "A survey of augmented reality," *Presence-Teleoperators and Virtual Environments*, **6**(4), pp. 355-385, 1997.

[3] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE Transactions on Information and Systems*, **E77-D**(12), pp. 1321-1329, 1994.

[4] M. Wakid, C. Kirmizibayrak, and J. K. Hahn, "Texture mapping volumes using GPU-based polygon-assisted raycasting," in *16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games*, (to appear 2011).

[5] Can Kirmizibayrak, Mike Wakid, Steve Bielamowicz, and J. Hahn, "An interactive multimodal visualization method for medical data exploration and image guided surgeries," presented at the 2011 Annual Meeting of the Society for Imaging Informatics in Medicine, 2011.

[6] Yeny Yim, Xuanyi Chen, Mike Wakid, Steve Bielamowicz, and J. Hahn, "2D-3D registration using gradient-based MI for image guided surgery systems," in *Proceedings of SPIE Medical Imaging 2011*, 2011.

[7] Yeny Yim, Mike Wakid, Can Kirmizibayrak, Steve Bielamowicz, and J. Hahn, "Registration of 3D CT data to 2D endoscopic image using a gradient mutual information based viewpoint matching for image-guided medialization laryngoplasty," *Journal of Computing Science and Engineering*, **4**(4), pp. 368-387, 2010.

[8] Can Kirmizibayrak, Mike Wakid, S. Bielamowicz, and J. Hahn, "Interactive visualization for image guided medialization laryngoplasty," in *Computer Graphics International 2010*, 2010, pp. 8-11.

[9] W. E. L. Grimson, G. J. Ettinger, S. J. White, T. Lozano-Perez, W. M. Wells Iii, and R. Kikinis, "An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization," *IEEE Transactions on Medical Imaging*, **15**(2), pp. 129-140, 1996.

[10] P. Paul, O. Fleig, and P. Jannin, "Augmented virtuality based on stereoscopic reconstruction in multimodal image-guided neurosurgery: Methods and performance evaluation," *IEEE Transactions on Medical Imaging*, **24**(11), pp. 1500-1511, 2005.

[11] H. H. S. Ip and L. Yin, "Constructing a 3D individualized head model from two orthogonal views," *The Visual Computer*, **12**(5), pp. 254-266, 1996.

[12] S. Lee, "Three-dimensional photography and its application to facial plastic surgery," *Archives of Facial Plastic Surgery*, **6**(6), pp. 410-414, 2004.

[13] R. M. Koch, S. H. M. Roth, M. H. Gross, A. P. Zimmermann, and H. F. Sailer, "A framework for facial surgery simulation," in *Proceedings of the 18th spring conference on Computer graphics*, 2002, pp. 33-42.

[14] G. R. J. Swennen, W. Mollemans, and F. Schutyser, "Three-dimensional treatment planning of orthognathic surgery in the era of virtual imaging," *Journal of Oral and Maxillofacial Surgery*, **67**(10), pp. 2080-2092, 2009.

[15] A. M. McCance, J. P. Moss, W. R. Wright, A. D. Linney, and D. R. James, "A three-dimensional soft tissue analysis of 16 skeletal class III patients following bimaxillary surgery," *British Journal of Oral and Maxillofacial Surgery*, **30**(4), pp. 221-232, 1992.

[16] J. P. Moss, A. M. McCance, W. R. Fright, A. D. Linney, and D. R. James, "A three-dimensional soft tissue analysis of fifteen patients with class II, division 1 malocclusions after bimaxillary surgery," *American Journal of Orthodontics and Dentofacial Orthopedics*, **105**(5), pp. 430-437, 1994.

[17] M. Y. Hajeer, A. F. Ayoub, D. T. Millett, M. Bock, and J. P. Siebert, "Three-dimensional imaging in orthognathic surgery: the clinical application of a new method," *The International Journal of Adult Orthodontics and Orthognathic Surgery*, **17**(4), pp. 318-330, 2002.

[18] M. Y. Hajeer, D. T. Millett, A. F. Ayoub, and J. P. Siebert, "Current products and practices: Applications of 3D imaging in orthodontics: Part I," *Journal of Orthodontics*, **31**(1), pp. 62-70, 2004.

[19] J. Xia, D. Wang, N. Samman, R. W. K. Yeung, and H. Tideman, "Computer-assisted three-dimensional surgical planning and simulation: 3D color facial model generation," *International Journal of Oral and Maxillofacial Surgery*, **29**(1), pp. 2-10, 2000.

[20] J. Xia, H. H. S. Ip, N. Samman, H. T. F. Wong, J. Gateno, W. Dongfeng, . . . H. Tideman, "Three-dimensional virtual-reality surgical planning and soft-tissue prediction for orthognathic surgery," *IEEE Transactions on Information Technology in Biomedicine* **5**(2), pp. 97-107, 2001.

[21] Y. Iwakiri, K. Yorioka, and T. Kaneko, "Fast texture mapping of photographs on a 3D facial model," in *Image and Vision Computing New Zealand 2003*, 2003, pp. 390-395.

[22] S. A. Schendel and C. Lane, "3D orthognathic surgery simulation using image fusion," *Seminars in Orthodontics*, **15**(1), pp. 48-56, 2009.

[23] J. P. Helferty, A. J. Sherbondy, A. P. Kiraly, and W. E. Higgins, "Computer-based system for the virtual-endoscopic guidance of bronchoscopy," *Computer Vision and Image Understanding*, **108**(1-2), pp. 171-187, 2007.

[24] W. E. Higgins, J. P. Helferty, K. Lu, S. A. Merritt, L. Rai, and K. C. Yu, "3D CT-video fusion for image-guided bronchoscopy," *Computerized Medical Imaging and Graphics*, **32**(3), pp. 159-173, 2008.

[25] L. G. Johnson, P. Edwards, and D. Hawkes, "Surface transparency makes stereo overlays unpredictable: The implications for augmented reality," *Studies in Health Technology and Informatics*, **94**, pp. 131-136, 2003.

[26] J. E. Swan II, A. Jones, E. Kolstad, M. A. Livingston, and H. S. Smallman, "Egocentric depth judgments in optical, see-through augmented reality," *IEEE Transactions on Visualization and Computer Graphics*, **13**(3), pp. 429-442, 2007.

[27] M. Lerotic, A. Chung, G. Mylonas, and G. Z. Yang, "pq-space based non-photorealistic rendering for augmented reality," *Medical Image Computing and Computer-Assisted Intervention*, pp. 102-109, 2007.

[28] F. Devernay, F. Mourgues, and È. Coste-Manière, "Towards endoscopic augmented reality for robotically assisted minimally invasive cardiac surgery," 2001, pp. 16-20.

[29] Quicktime VR, available: http://www.apple.com/quicktime/, "site accessed on April 25, 2011".

[30] Surround Video, available: http://www.immersivemedia.com, "site accessed on April 25, 2011".

[31] A. Behrens, "Creating panoramic images for bladder fluorescence endoscopy," *Acta Polytechnica Journal of Advanced Engineering*, **48**(3), pp. 50-54, 2008.

[32] W. Konen, B. Breiderhoff, and M. Scholz, "Real-time image mosaic for endoscopic video sequences," *Bildverarbeitung für die Medizin 2007*, pp. 298-302, 2007.

[33] S. Seshamani, W. Lau, and G. Hager, "Real-time endoscopic mosaicking," *Lecture Notes in Computer Science*, **4190**, pp. 355, 2006.

[34] R. E. Carroll and S. M. Seitz, "Rectified surface mosaics," *International Journal of Computer Vision*, **85**(3), pp. 307-315, 2009.

[35] D. Koppel, Y. F. Wang, and H. Lee, "Image-based rendering and modeling in video-endoscopy," *IEEE International Symposium on Biomedical Imaging: Macro to Nano*, **1**, pp. 269-272, 2004.

[36] T. Bergen, S. Ruthotto, C. Munzenmayer, S. Rupp, D. Paulus, and C. Winter, "Feature-based real-time endoscopic mosaicking," in *6th International Symposium on Image and Signal Processing and Analysis*, 2009, pp. 695-700.

[37] B. André, T. Vercauteren, A. M. Buchner, M. B. Wallace, and N. Ayache, "Endomicroscopic video retrieval using mosaicing and visualwords," in *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 2010, pp. 1419-1422.

[38] D. Dey, P. Slomka, D. Gobbi, and T. Peters, "Mixed reality merging of endoscopic images and 3-D surfaces," in *Proceedings of MICCAI*, 2000, pp. 796-803.

[39] D. Dey, D. G. Gobbi, K. J. M. Surry, P. J. Slomka, and T. M. Peters, "Mapping of endoscopic images to object surfaces via ray-traced texture mapping for image guidance in neurosurgery," in *Medical Imaging 2000*, 2000, pp. 290–300.

[40] D. Dey, D. G. Gobbi, P. J. Slomka, K. J. M. Surry, and T. M. Peters, "Automatic fusion of freehand endoscopic brain images to three-dimensional surfaces: Creating stereoscopic panoramas," *IEEE Transactions on Medical Imaging*, **21**(1), pp. 23-30, 2002.

[41] L. Rai and W. E. Higgins, "Image-based rendering method for mapping endoscopic video onto CT-based endoluminal views," in *Proceedings of SPIE 6141*, 2006, pp. 1-12.

[42] X. Wang, Q. Zhang, Q. Han, R. Yang, M. Carswell, B. Seales, and E. Sutton, "Endoscopic video texture mapping on pre-built 3-D anatomical objects without camera tracking," *Medical Imaging, IEEE Transactions on*, **29**(6), pp. 1213-1223, 2010.

[43] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, **21**(4), pp. 163-169, 1987.

[44] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross, "Real-time ray-casting and advanced shading of discrete isosurfaces," *Computer Graphics Forum*, **24**(3), pp. 303–312, 2005.

[45] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, 2001, pp. 9-16.

[46] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, **8**(3), pp. 29-37, 1988.

[47] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Proceedings of SIGGRAPH '88*, 1988, pp. 65-74.

[48] M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, **9**(3), pp. 245-261, 1990.

[49] T. J. Cullip and U. Neumann, "Accelerating volume reconstruction with 3D texture hardware," Technical report TR93-027, University of North Carolina, Chapel Hill, 1994.

[50] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proceedings of the 1994 symposium on Volume visualization*, 1994, pp. 91-98.

[51] J. Krüger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proceedings IEEE Visualization 2003*, 2003, pp. 287-292.

[52] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, and K. Bühler, "Perspective isosurface and direct volume rendering for virtual endoscopy applications," in *Proceedings of Eurovis*, 2006, pp. 315-322

[53] W. Hong, F. Qiu, and A. Kaufman, "GPU-based object-order ray-casting for large datasets," in *Fourth International Workshop on Volume Graphics*, 2005, pp. 177-240.

[54] R. S. Avila, L. M. Sobierajski, and A. E. Kaufman, "Towards a comprehensive volume visualization system," in *Proceedings of the 3rd conference on Visualization '92*, 1992, pp. 13-20.

[55] W. Leung, N. Neophytou, and K. Mueller, "SIMD-aware ray-casting," in *Volume Graphics*, 2006, pp. 59-62.

[56] J. Mensmann, T. Ropinski, and K. Hinrichs, "Accelerating volume raycasting using occlusion frustums," in *Eurographics/IEEE 7th International Symposium on Volume and Point-Based Graphics*, 2008, pp. 147-154.

[57] V. Vidal, X. Mei, and P. Decaudin, "Simple empty-space removal for interactive volume rendering," *Journal of Graphics, GPU, and Game Tools*, **13**(2), pp. 21-36, 2008.

[58] B. Liu, G. J. Clapworthy, and F. Dong, "Accelerating volume raycasting using proxy spheres," *Computer Graphics Forum*, **28**(3), pp. 839-846, 2009.

[59] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide*, Fifth ed.: Addison-Wesley, 2005.

[60] P. S. Heckbert, "Survey of texture mapping," *IEEE Computer Graphics and Applications*, **6**(11), pp. 56-67, 1986.

[61] E. A. Bier and K. R. Sloan, "Two-part texture mappings," *IEEE Computer Graphics and Applications* **6**(9), pp. 40-53, 1986.

[62] O. Shibolet and D. Cohen-Or, "Coloring voxel-based objects for virtual endoscopy," in *Proceedings of the 1998 IEEE symposium on Volume visualization*, 1998, pp. 15-22.

[63] P. Shen and P. Willis, "Texture for volume character animation," in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia 2005*, 2005, pp. 255-264.

[64] K. Bürger, J. Krüger, and R. Westermann, "Direct volume editing," *IEEE Transactions on Visualization and Computer Graphics*, **14**(6), pp. 1388-1395, 2008.

[65] D. Benson and J. Davis, "Octree textures," *ACM Transactions on Graphics (TOG)*, **21**(3), pp. 785-790, 2002.

[66] D. DeBry, J. Gibbs, D. D. L. Petty, and N. Robins, "Painting and rendering textures on unparameterized models," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 763-768.

[67] S. Lefebvre, S. Hornus, and F. Neyret, "Octree textures on the GPU," *GPU gems*, **2**, pp. 595–613, 2005.

[68] A. E. Lefohn, J. Kniss, R. Strzodka, S. Sengupta, and J. D. Owens, "Glift: Generic, efficient, random-access GPU data structures," *ACM Transactions on Graphics (TOG)*, **25**(1), pp. 60-99, 2006.

[69] K. Buerger, J. Kruger, and R. Westermann, "Sample-based surface coloring," *IEEE Transactions on Visualization and Computer Graphics*, **16**(5), pp. 763-776, 2010.

[70] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information," *International Journal of Computer Vision*, **24**(2), pp. 137-154, 1997.

[71] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(2), pp. 239-256, 1992.

[72] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation*, **3**(4), pp. 323-344, 1987.

[73] H. K. Pedersen, "Decorating implicit surfaces," in *Proceedings of SIGGRAPH '95*, 1995, pp. 291-300.

[74] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *Proceedings of SIGGRAPH '96*, 1996, pp. 313-324.

[75] J. Maillot, H. Yahia, and A. Verroust, "Interactive texture mapping," in *Proceedings of SIGGRAPH '93*, 1993, pp. 27-34.

[76] I. Guskov, K. Vidim e, W. Sweldens, and P. Schröder, "Normal meshes," in *Proceedings of SIGGRAPH '00*, 2000, pp. 95-102.

[77] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, "MAPS: Multiresolution adaptive parameterization of surfaces," in *Proceedings of SIGGRAPH '98*, 1998, pp. 95-104.

[78] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, "Texture mapping progressive meshes," in *Proceedings of SIGGRAPH '01*, 2001, pp. 409-416.

[79] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution analysis of arbitrary meshes," in *Proceedings of SIGGRAPH '95*, 1995, pp. 173-182.

[80] P. Cignoni, C. Montani, R. Scopigno, and C. Rocchini, "A general method for preserving attribute values on simplified meshes," in *Proceedings of IEEE Visualization Conference*, 1998, pp. 59-66.

[81] V. Milenkovic, "Rotational polygon containment and minimum enclosure using only robust 2D constructions," *Computational Geometry*, **13**(1), pp. 3-19, 1999.

[82] B. Lévy, S. Petitjean, N. Ray, and J. Maillot, "Least squares conformal maps for automatic texture atlas generation," *ACM Transactions on Graphics*, **21**(3), pp. 362-371, 2002.

[83] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-Time Volume Graphics*: A K Peters, 2006.

[84] K. Myers and L. Bavoil, "Stencil routed a-buffer," *ACM SIGGRAPH 2007 Technical Sketch Program*, 2007.

[85] C. Everitt, "Order-independent transparency," Technical report, NVIDIA Corporation, 2001.

[86] L. Bavoil and K. Myers, "Order independent transparency with dual depth peeling," *NVIDIA OpenGL SDK*, 2008.

[87] J. Danskin and P. Hanrahan, "Fast algorithms for volume ray tracing," in *ACM Workshop on Volume Visualization '92*, 1992, pp. 91-98.

[88] R. Yagel and Z. Shi, "Accelerating volume animation by space-leaping," in *Proceedings of IEEE Visualization '93*, 1993, pp. 62-69.

[89] J. Freund and K. Sloan, "Accelerated volume rendering using homogeneous region encoding," in *Proceedings of IEEE Visualization '97*, 1997, pp. 191-197.

[90] J. Krüger, J. Schneider, and R. Westermann, "Clearview: An interactive context preserving hotspot visualization technique," *IEEE Transactions on Visualization and Computer Graphics*, **12**(5), pp. 941-948, 2006.

[91] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Transactions on Graphics*, **2**(4), pp. 217-236, 1983.