An Interactive Virtual Endoscopy Tool with Automatic Path Generation

by

Delphine Nain

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2002

Author ....
Department of Electrical Engineering and Computer Science
May 10, 2002

Certified by ....
W. Eric L. Grimson
Bernard Gordon Professor of Medical Engineering
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

An Interactive Virtual Endoscopy Tool with Automatic Path Generation

by

Delphine Nain

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 2002, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

In this thesis, we present an integrated virtual endoscopy (VE) software package. We describe its system architecture as well as two novel real-time algorithms integrated in the suite. The first algorithm automatically extracts in real-time a centered trajectory inside a 3D anatomical model between two user-defined points. The second algorithm uses dynamic programming to align centerlines from models of the same anatomical structure subject to a non-linear deformation. This allows for synchronized fly-throughs of non-linearly deformed intra-patient datasets.

We discuss the wide range of applications of the VE suite, including surgical planning, 3D model analysis, medical training and post-treatment analysis. The VE suite uniquely integrates four important features of virtual endoscopy in a single environment. It provides capabilities for interactive navigation, fusion of 3D surface and volume information, automatic and real-time fly-through generation and synchronized fly-throughs of several datasets acquired from the same patient in different body positions. Our tool has been evaluated by medical students and doctors at the Brigham and Women's hospital in various clinical cases.

Thesis Supervisor: W. Eric L. Grimson
Title: Bernard Gordon Professor of Medical Engineering

Acknowledgments

I first learned about the field of Medical Imaging at the 1999 SIGGRAPH conference while attending a seminar chaired by Dr. Ron Kikinis. I admired the impact of the research presented at the seminar, as well as its real inter-disciplinary nature: doctors and engineers had learned to talk a common language to design medical applications that enhanced the vision of doctors by using the knowledge in the fields of computer graphics, signal processing and artificial intelligence.

I am very grateful to Professor Eric Grimson for letting me join the Medical Vision group at the Artificial Intelligence (AI) Laboratory of MIT as an undergraduate researcher in the Spring of 2000 and supporting my work for the past two years. The excellence in research and friendly atmosphere of the Vision group reflect his incredible leadership and his dedication to his students. The work in this thesis would not have been possible without his constant support and motivation.

Our group at the MIT AI Lab has been collaborating closely for several years with the Surgical Planning Laboratory (SPL) of Brigham and Women's Hospital. I am very grateful to Dr. Ron Kikinis and Dr. Ferenc Jolesz for welcoming me to the SPL. Their pioneering vision has made the SPL one of the world's leading medical imaging lab where doctors and scientists interact on a daily basis. Dr Kikinis suggested that I work on virtual endoscopy and he has been my greatest supporter since the beginning and has really shaped my research with his knowledge of both the medical sciences and computer science.

I felt really welcomed and encouraged by everybody in the AI Lab Vision group and the SPL. They are an amazing group of researchers with an incredible knowledge and a contagious enthusiasm with whom I have spent some of the best times of my life while discussing research, skiing in Vermont or traveling to Baltimore and Utrecht. I would especially like to thank Lauren O'Donnell for being such a great mentor and friend in the past two years, teaching me everything from the bases of 3D Slicer to the intricate moves of Flamenco. I would also like to thank Sandy Wells, David Gering, Polina Golland, Samson Timoner, Eric Cosman, Lilla Zollei, John Fisher, Kilian Pohl and many others in our group for always being available to answer my questions and share their wisdom and knowledge. The work in synchronized virtual colonoscopy presented in this thesis was inspired by previous work by Sandy and Eric and I am very thankful for our brainstorming sessions. I would also like to thank alumni of our group who have been a great source of inspiration for me: Tina Kapur, Michael Leventon and Liana Lorigo.

At the SPL, I had the privilege to collaborate with Steven Haker and Carl-Fredrik Westin who helped shape my work on centerline extraction and synchronized virtual colonoscopy. I will keep fond memories of our research brainstorms in Au Bon Pain during half-off baking sale! I would like to thank Peter Everett for all his help in the design of the Virtual Endoscopy Tool and the "SPL gang" for making this past year so much fun: Hanifa Dostmohamed , Karl Krissian, Raul San Jose Estepar, Soenke "Luigi" Bartling, Mike Halle and Florent Thalos. I would also like to acknowledge the work and support of many SPL medical collaborators who tested and used the Virtual Endoscopy Tool and helped in its design: Dr. Bonglin Chung, Soenke Bartling, Dr. Lennox Hoyte, Dr. Hoon Ji. Thank you for leading the way in clinical research that involves medical imaging tools.

I met many people around the world who inspired my research and introduced me to fascinating topics in medical imaging: Allen Tannenbaum and Anthony Yezzi from Georgia Tech, Olivier Faugeras, Nicolas Ayache, Herve Delingette, and Eve Coste-Maniere from INRIA in France, Thomas Deschamps from Philips Medical Imaging and Universite de Paris, Olivier Cuisenaire from L'Ecole Polytechnique Federale de Lausanne. Thomas Deschamps and Olivier Cuisenaire both worked on an aspect of centerline extraction for their PhD thesis and I would like to thank them for our fruitful discussions and their useful suggestions for my work on central path planning.

Finally, I would like to thank those who have always given me unconditional moral and emotional support: my boyfriend Omar, my parents Philippe and Marie-Pierre, my family and all my friends in France and in United States.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we present the design and implementation of a 3D Virtual Endoscopy software program for facilitating diagnostic and surgical planning phases of endoscopic procedures.

Our system allows the user to interactively and intuitively explore 3D patient-specific anatomical models and create and update a fly-through trajectory through models of any topology to simulate endoscopy. By fusing information about the surface being observed and the original grayscale data obtained from a medical scan, our tool augments the human eye with an enhanced vision of the anatomy.

To create a fly-through in minimal time, our system contains an automatic path planning algorithm that extracts in real-time a centerline from a 3D surface model, between two user-defined points. The centerline is then used like a virtual rail to guide the virtual endoscope during the fly-through exploration. The user still has complete control of the endoscope during a fly-through and can move it away from its trajectory to explore the surface more closely, as well as modify the trajectory path to go through manually defined points. This combination of fast automatic path extraction and precise manual control of the endoscope provides an improved system over current virtual endoscopy programs used clinically that do not give the user a full and intuitive control of the virtual endoscope.

Our system also contains an advanced feature that allows users to synchronize fly-throughs inside different datasets of the same organ scanned in different positions and therefore non-linearly deformed with respect to each other. This synchronization allows the user to cross-correlate the registered datasets and can be a valuable tool for diagnosis and post-treatment analysis.

We have integrated our program in the 3D Slicer, a medical visualization program created at the MIT AI Lab in collaboration with the Surgical Planning Laboratory at the Brigham and Women's Hospital [1]. It provides surgeons with a single environment to generate 3D models, use quantitative analysis tools and conduct a virtual endoscopy. The Virtual Endoscopy program has been used for virtual colonoscopy, virtual cystoscopy, auricular and cardiovascular projects.

1.1 Background

Cancer[1] is the second highest cause of death in the United States (23.0 % of US deaths in 1999). Men have a 43.8 % risk of being diagnosed with cancer in their lifetime, women have a 38.4 % risk [2]. If diagnosed and treated early, some cancers can be cured or cancer cells can be removed and limited in their spread, which increases the chances of survival for the patient [3].

Some cancers, such as skin cancer (melanoma) or breast cancer, can be detected with the eye or with a physical examination . Unfortunately, over 80% of cancers occur in internal organs and cannot be detected or diagnosed without a way to explore the *inside* of the body. For this purpose, techniques such as medical imaging and endoscopy were developed to observe internal organs. We present these techniques in detail in the next two sections. Medical scans and endoscopy can also be used to detect other forms of pathologies in the body, such as ulcers, hernias, inflammations, stenosis or malformations which also affect a large portion of the population and can need medical treatment.

Medical scans and endoscopy are often complimentary diagnosis tools since both these techniques have their advantages and limitations. Virtual endoscopy has been

[1]Cancer is the uncontrolled growth of abnormal cells which have mutated from normal tissues. Cancer can kill when these cells prevent normal function of affected vital organs or spread throughout the body to damage other key systems.

proposed recently to unite these techniques and leverage their strength, as well as bring its own unique advantages in four important areas: diagnosis, surgical planning, measurement and analysis, and post-treatment.

In the next sections we explain in more details all the conventional diagnostic options and then present virtual endoscopy as an alternative.

1.2 Conventional Endoscopy

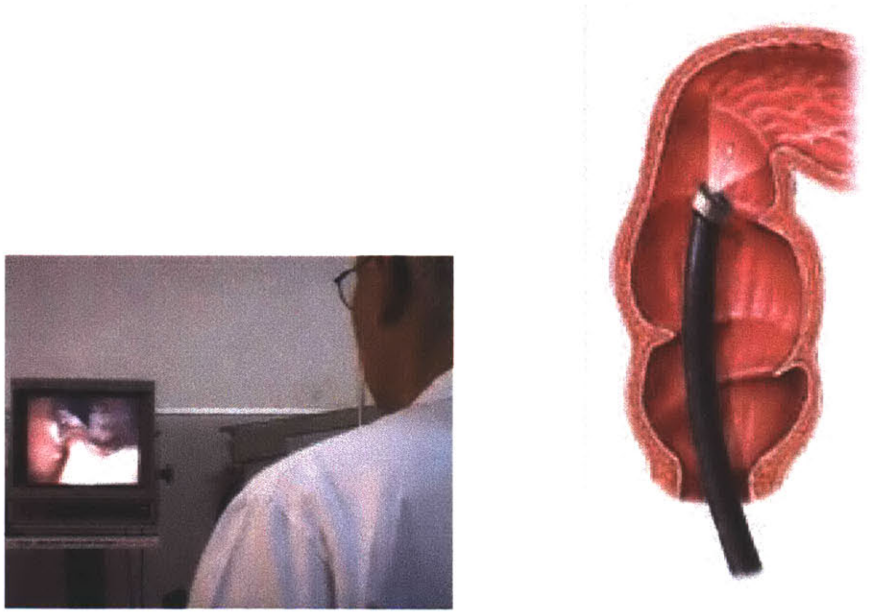


Figure 1-1: Conventional Endoscopy: on left image, the clinician can view the inside of a hollow organ on a video screen by inserting an endoscope inside the organ. The endoscope, seen on the right image is a video camera at the end of a flexible guided tube.

By using an endoscope, a device consisting of a tube and an optical system, the operator can view the inner surface of the organ using video-assisted technology (Figure 1-1) and he can change the position and the angle of the probe to conduct a full exploration of the organ. Endoscopy can be entirely diagnostic (detection of pathologies such as tumors, ulcers, cysts) as well as an actual therapeutic procedure where a tool can be added to the tip of the endoscope to ablate or cauterize a pathology, open obstructions or treat bleeding lesions.

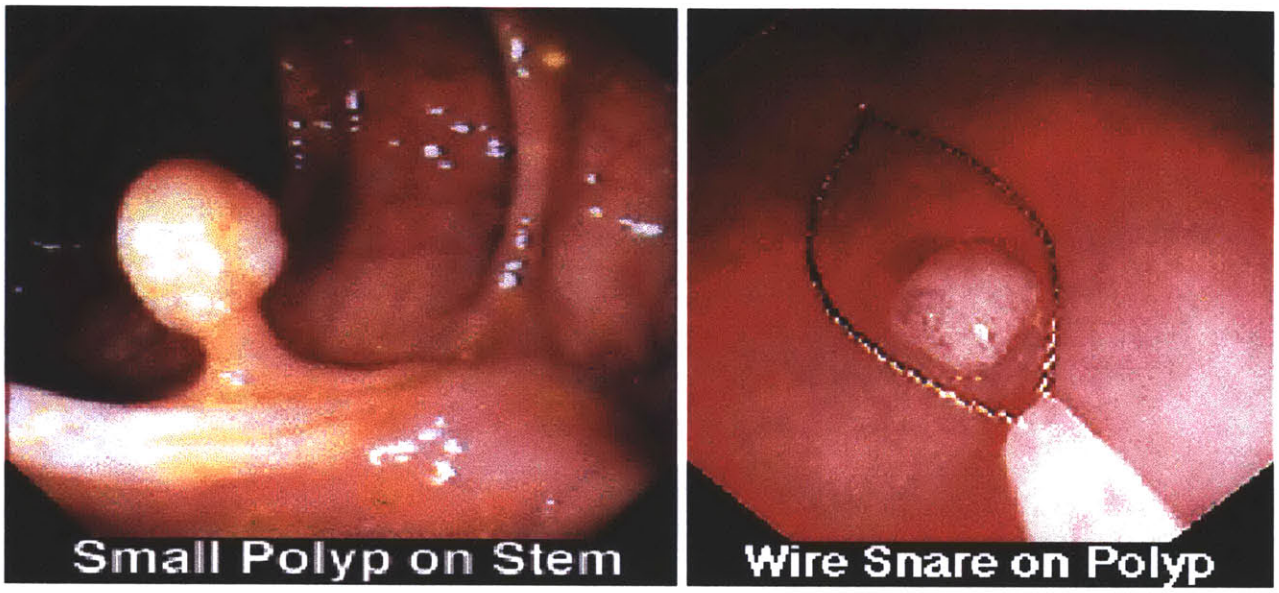


Figure 1-2: The left image shows a snapshot taken by the endoscope inside the colon of a patient. The protruding bump attached on a pedicle is a growth of tissue called a polyp that could potentially cause cancer. The right image shows a snare tool attached to the tip of the endoscope that is used to remove the polyp.

Common endoscopic procedures include:

colonoscopy : the procedure of viewing the interior lining of the large intestine (colon) using a colonoscope. This procedure is most commonly performed to detect polyps, tumors that are an outgrowth of normal or abnormal tissue that may be attached by a pedicle [4], as seen on the left image of Figure 1-2. If a polyp is detected, a sample of it can be extracted with a tool added at the tip of the endoscope to conduct a biopsy to determine if the polyp is benign or if its growth is malignant and therefore might cancer. If the polyp is malignant, it can be removed during the endoscopy as seen in the right image of Figure 1-2. Colonoscopies can also be performed to evaluate unexplained abdominal pain, to determine the type and extent of inflammatory bowel disease and to monitor people with previous polyps, colon cancer, or a family history of colon cancer.

cystoscopy : a procedure that enables the urologist to view the inside of the bladder in great detail using a specialized endoscope called a cystoscope. The urologist typically looks for abnormalities such as tumors and polyps, or cysts, ulcers and bladder stones that can cause pain and infection.

laparoscopy : a procedure that allows the physician to look directly at the contents of the abdomen and pelvis. The purpose of this examination is to directly assess the presence of inflammation of the gallbladder (cholecystitis), appendix (appendicitis), and pelvic organs (Pelvic Inflammatory Disease) or to remove tumors before large operations such as liver and pancreatic resections.

esophagogastroduodenoscopy (EGD) : a procedure to examine the lining of the esophagus and stomach to look for tumors, ulcers, inflammations, strictures, obstructions or esophagus rings that can cause bleeding, swallowing difficulties and pain.

In all these procedures, the physician detects abnormalities by examining the wall appearance (color, texture) and its shape (presence of abnormal bumps or topologies). One limitation of this method is the inability to assess the extent of lesions beyond the wall of the organ and localize the lesion relative to surrounding anatomic structures, since the endoscope is restricted in its exploration to the inside of the organ [5].

1.3 Medical Scans

Medical scanners, such as Computed Tomography (CT) or Magnetic Resonance (MR) scanners, output data in the form of 3D arrays of volume elements called *voxels*. Each voxel is a scalar value representative of a physical property to which the scanner is sensitive. For example, CT images are a map of X-ray attenuation coefficients through the body and are therefore suited to image tissue/air/bone contrast. MR images show a radio-frequency signal emitted by hydrogen atoms when they relax back to their equilibrium positions after having been reoriented by a strong magnetic field (typically 1.5 Tesla). MR imaging is particularly suited to image soft tissue such as brain tissue. Volumetric data obtained from the scanners is stored as a stack of 2D images (Figure 1-3), often called *slices*.

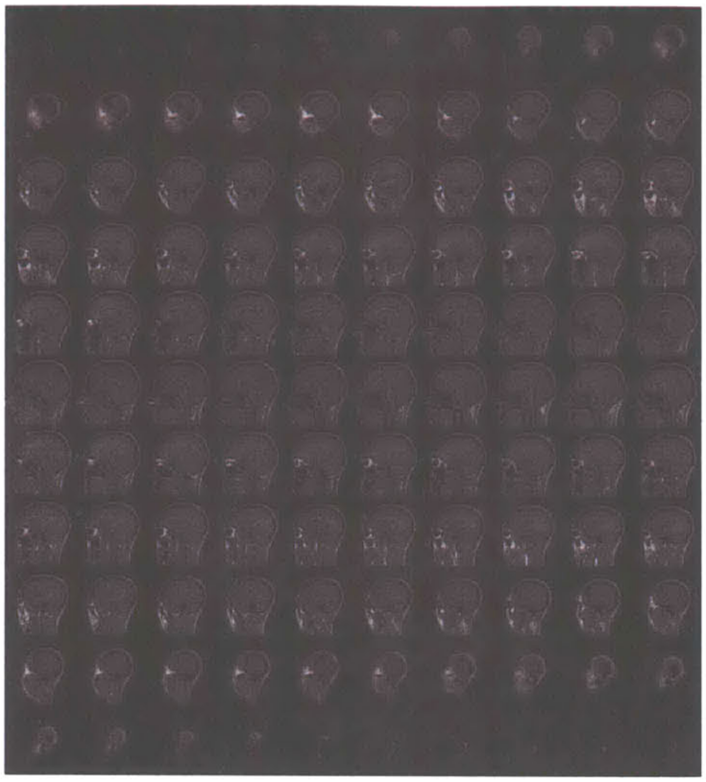


Figure 1-3: A 3D volume from a medical scanner is stored as a set of 2D images

Because medical scans image cross sections of the body, they contain information not available to the endoscope, such as information on lesion tissue shape through and beyond the walls of the organ. Another advantage of the medical scanning technique is that it is non-invasive and causes much less discomfort and pain to the patient compared to an endoscopic procedure.

However, one drawback of this method is that the surgeon has to mentally align and combine the contiguous slices in order to "see" in 3D and perform a diagnosis. Physicians are trained to be able to perform a diagnosis by recognizing abnormalities on scans but, in the words of Dr. Lennox Hoyte, a surgeon in the department of Obstetrics and Gynecology at the Brigham and Womens, they do not have a good understanding of the 3D structures due to the nature of the 2D planes used to visualize the information. Dr. Hoyte observed that most surgeons were surprised to see the shape of 3D models of the pelvic organs of a living person reconstructed from a medical scan since they do not usually have access to that information (even during surgery most organs are hidden by other ones).

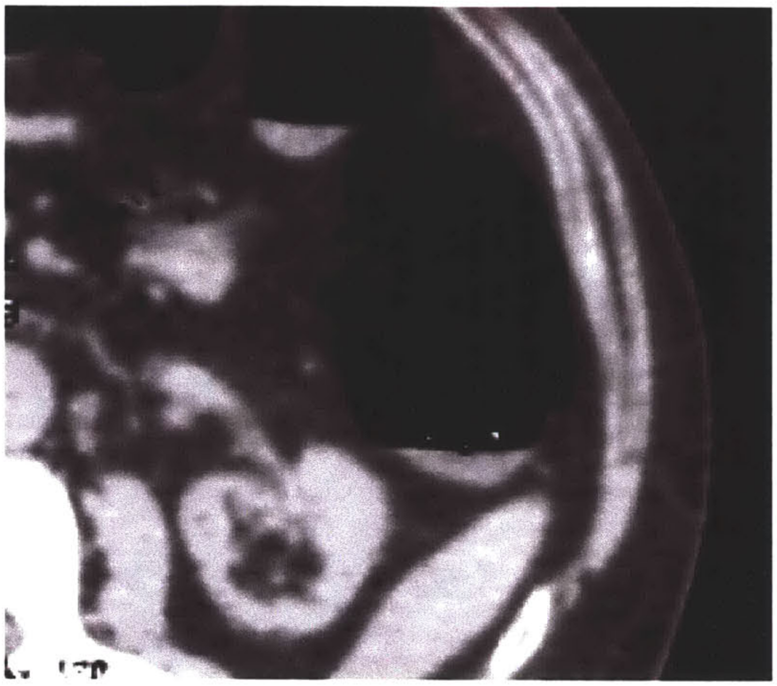


Figure 1-4: Retained fluid seen on an axial CT slice. The fluid can be seen in white inside the colon that is filled with air and therefore black.

Another drawback is that the resolution of the images is much lower than the video image seen during a real endoscopy which can cause some artifacts to be misclassified. In some cases, the presence of material such as fluid may be misclassified as tissue. Particularly in the case of CT scans of the colon, the presence of fake polyps, material left in the colon which may adhere to the colon wall and appear much like a polyp, can make the task of finding true polyps difficult. Figure 1-4 shows a slice through the CT scan of a colon where fluid retention can be seen, another factor complicating the exam.

In order to better differentiate actual polyps from artifacts, and to better view the inner organ wall surface in the presence of fluid, it is common practice to obtain two CT scans of the patient, one where the patient lies on his stomach (prone position) and another one where the patient lies on his back (supine position). Fluid in the colon will naturally appear on the anterior colon wall when the patient is in the prone position, and on the posterior wall when in the supine. Artifacts may also change position between the two scans, allowing the radiologist to differentiate these

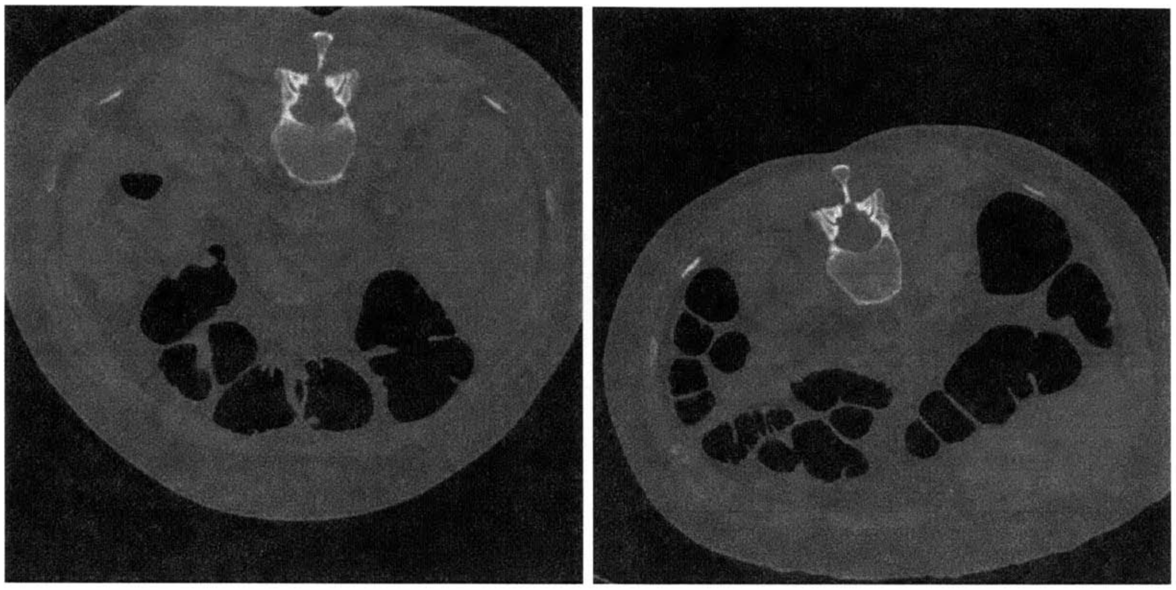


Figure 1-5: Axial slices through supine (left) and prone (right) scans. Although the same vertebra is pictured, the colon and other anatomical structures are not aligned.

structures from true polyps [6]. However, comparing findings in both supine and prone scans is a challenge since the colon changes orientation, position and shape between the two scans due to gravity and pressure forces from other organs. Figure 1-5 shows an axial slice from both the supine and prone scans at the same body height (the third spinal vertebra). As can be seen, the colons in both scans are not aligned. Since the colon moved and rotated inside the body between the scan acquisitions, but the scanner still acquired cross-sectional slices at the same position and orientation through the body, corresponding slices in both scans show dramatically different cross sections of the colon.

In practice, the radiologist can attempt a manual registration by using anatomical landmarks such as spinal vertebrae to observe images through similar axial planes, and then scroll through adjacent slices to try to find similar structures in the colon wall. Such methods however, are difficult, inaccurate and time consuming. We present a solution to this problem with our VE software, as described in chapter 4.

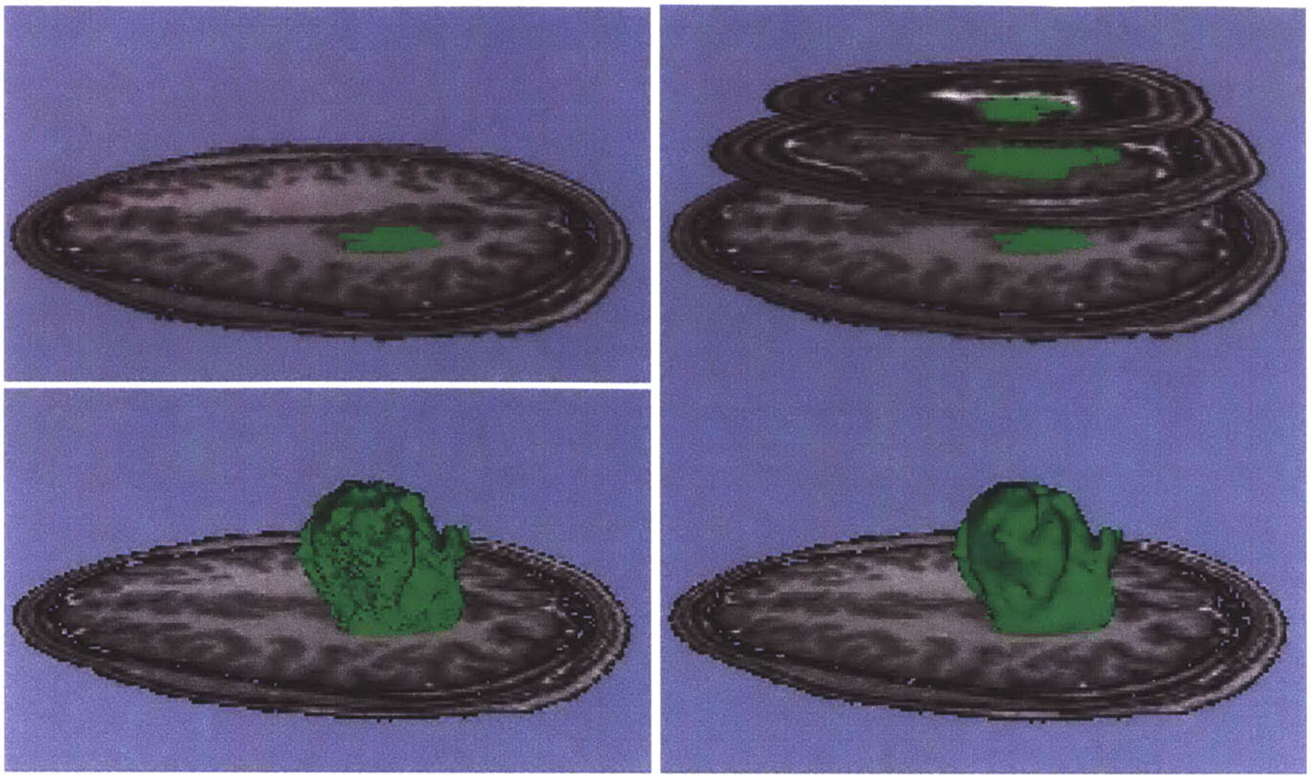


Figure 1-6: The label map of a tumor is created by segmenting slices (top). Next, a polygonal surface is created to encompass the segmentation (bottom-left). This surface is smoothed (bottom-right) to remove the digitization artifacts.

1.4 3D Visualization

To analyze medical imagery, it is often valuable to create 3-dimensional patient-specific models from the original CT or MRI volume, as described in the previous section. The first step in generating a 3D model is segmentation of anatomical structures on the slices of the acquired medical scans. Segmentation is the process of labeling pixels that belong to the same anatomical structure. For example in Figure 1-6, a tumor is outlined in green. Segmentation is usually done manually slice by slice, but its automation is an active area of research and many algorithms now exist to segment specific anatomical structures semi-automatically or automatically. Once the contour of a structure is labeled, a 3D triangulated surface model of the segmented anatomical structure can be created using algorithms such as the Marching Cubes algorithm [7]. The algorithm smoothly connects all the labeled pixels into one surface made of triangles, often called a "mesh". This surface can then be rendered onto the screen and colored and lighted just like any CAD model. The bottom two images of Figure 1-6 show the resulting 3D model of the tumor created by the Marching Cubes algorithm and then smoothed.

Computer programs enable the visualization and editing of 3D medical models. Examples of such software include the 3D Slicer developed at the MIT AI Lab [1] or the ANALYZE software developed at the Mayo Clinic [8]. The user can change the point of view by rotating around the model or zooming in to get a close-up view of the outer surface. The user can also zoom inside the model and rotate the view to explore the inner surface. However, the exploration is completely manual and can be time-consuming and frustrating to novice users who do not have experience navigating in a 3D screen. Furthermore, the exploration path cannot be saved, so the user has to reproduce it at every new session. Thus one of our goals was is to improve an existing medical visualization software, the 3D Slicer, by building a virtual endoscopy module that makes the exploration of 3D models faster, more intuitive and informative for surgical planning and data analysis.

1.5 Virtual Endoscopy

Virtual endoscopy is a general term that means exploring 3D virtual models. A virtual endoscopy system takes as input 3D reconstructed anatomical models and provides a technique to interactively explore the inner and outer surface of a model by giving the user control over a virtual endoscope that "films" what it sees and outputs it to a second screen. To introduce the field of virtual endoscopy, we first introduce three example case studies and then give an overview of our Virtual Endoscopy (VE) program and the advantages of virtual endoscopy.

1.5.1 Case Studies

The first example is shown in Figure 1-7. The virtual endoscope is represented on the left screen as a camera that has been placed inside the reconstructed 3D model

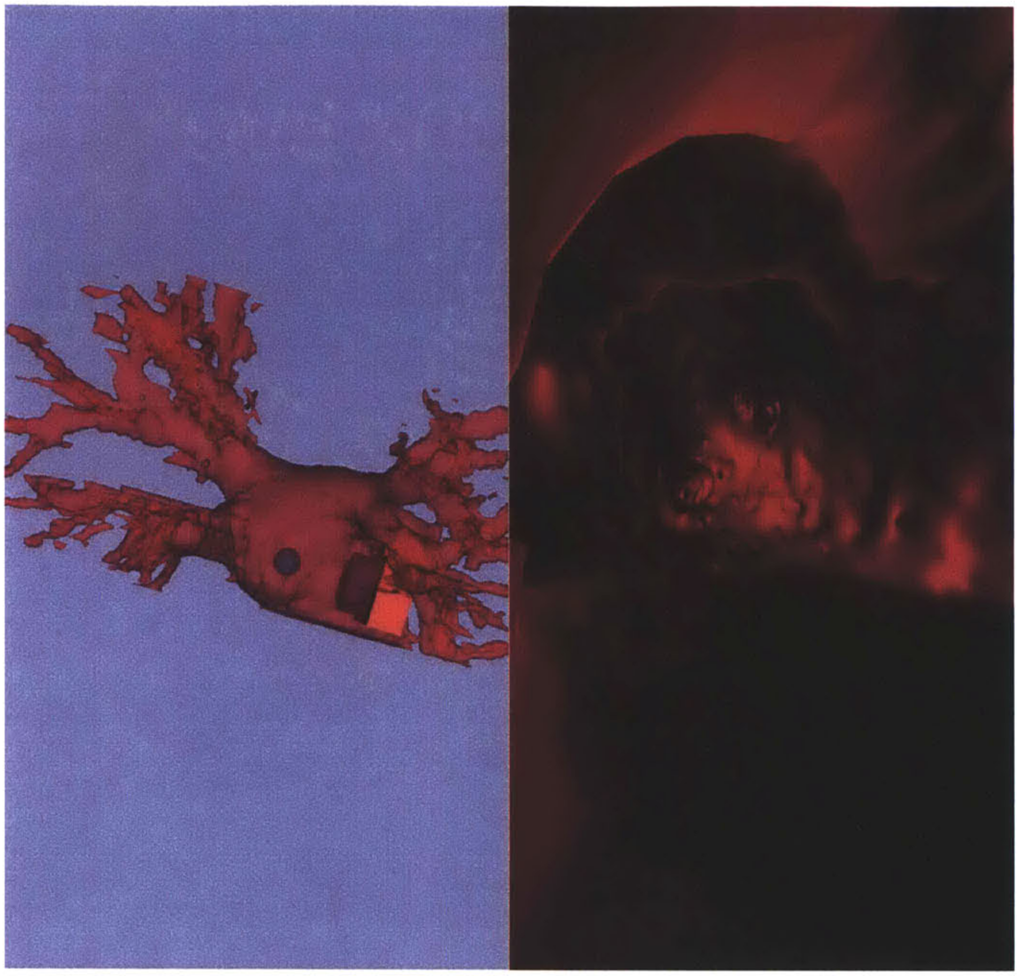


Figure 1-7: The left screen show the endoscope inside a reconstructed 3D model of a patient's heart cage looking down the right pulmonary vein. The right screen shows the atrium seen from the point of view of the endoscope.

of a 34 year old male patient's heart chamber. Specifically, the endoscope is placed to look down the right pulmonary veins, the veins responsible for carrying blood into the heart. By placing the virtual endoscope at specific positions, the physician is able to conduct measurements about the diameter of the vein for later analysis. If only the right screen were shown, just like in a conventional visualization system, the user could easily be disoriented and not be able to ascertain the position of the endoscope inside that model. What direction is the endoscope facing? How far away is it from the entrance point? By showing where the endoscope is located and what direction it is facing on the left screen, we allow the user to navigate in space and interpret the information seen inside the model in a global context.

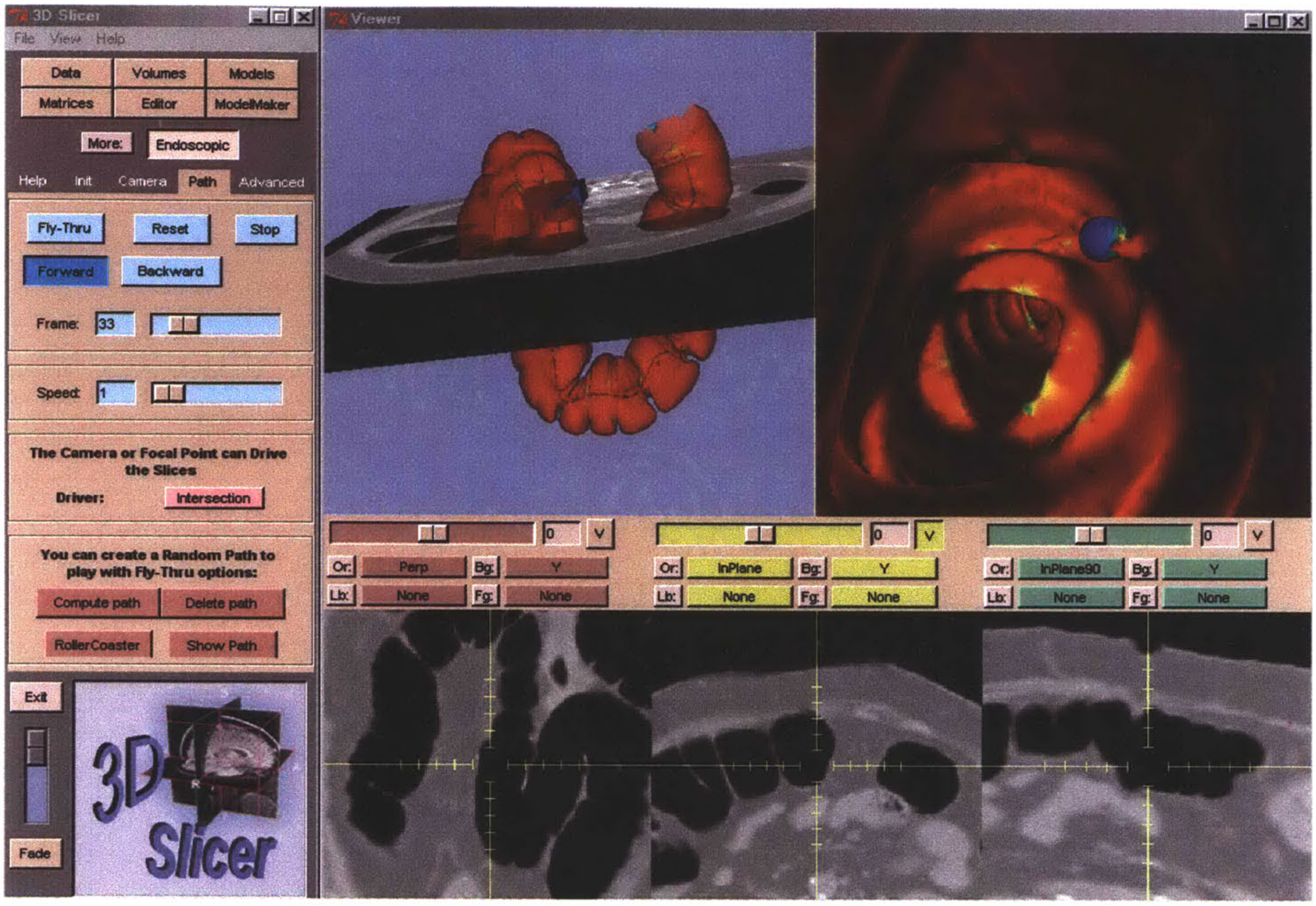


Figure 1-8: A virtual colonoscopy where a reformatted slice is shown orthogonal to the view plane of the endoscope. A polyp can be detected in the endoscopic view (right screen). The colon is colored based on curvature where blue represents a high curvature.

The second example is show in Figure 1-8. The endoscope is placed inside the reconstructed 3D model of the colon of a 66 year old female patient. The centerline shown in the colon was extracted automatically with the central path planning algorithm that we designed and integrated in our software. The centerline is used as a "rail" to guide the virtual endoscope. We refer to the process of flying the endoscope on that rail as a "fly-through".

Since the model is reconstructed from a medical scan, any abnormal shape of the colon surface appears on the scan and therefore appears as well on the virtual colon model. Here, the colon surface is colored based on curvature so that protruding bumps, like polyps, can be identified more easily. At any time during the fly-through, the user can stop the motion of the endoscope, for example when a suspicious area is detected and measurements need to be conducted. On the right screen of Figure 1-8, a polyp can be seen in the upper right corner (in blue) and measurement tags can be added to measure the size of the polyp.

Another interesting feature of our software is the fusion of 3D and 2D grayscale data. In the upper left screen of figure 1-8 the 3D data is the colon model in orange and the 2D data is a plane parallel to the endoscope view. On this plane, called a *reformatted slice*, the grayscale data are displayed from the original CT scan. Intuitively, it is as if we selected only the voxels that are intersected by that plane and displayed them on the screen. If the plane is oblique, the grayscale value of each pixel on the slice is assigned by interpolating the grayscale value of all the intersected voxels.

As the endoscope moves and gets re-oriented, the reformatted slice is updated to stay parallel to the endoscope view and the new grayscale information is retrieved from the original scan. There are many options to specify the orientation and position of the slice, and in this case the user centered the slice at the polyp seen on the right screen by clicking on it with the mouse. The three grayscale windows at the bottom of Figure 1-8, the *slice windows*, show reformatted planes parallel to each axis of the camera. The middle slice window is the one shown in the 3D screen (top left). The left and right windows show two other reformatted slices orthogonal to the middle one. With the reformatted slices, the physician can track the location of the polyp at different angle in the volume, view its extent beyond the wall of the colon and localize the polyp relative to other structures shown in the grayscale image, but not in the 3D view.

The third case study was an experiment to determine how well anatomical structures, lesions, and implants in the temporal (ear) bone are visualized using a very high resolution, Volume-CT (VCT) scanner as compared with a multislice-CT (MSCT) scanner [9]. Five temporal bone specimens were scanned both in the experimental VCT scanner and a state of the art MSCT. The individual 2D slices from both

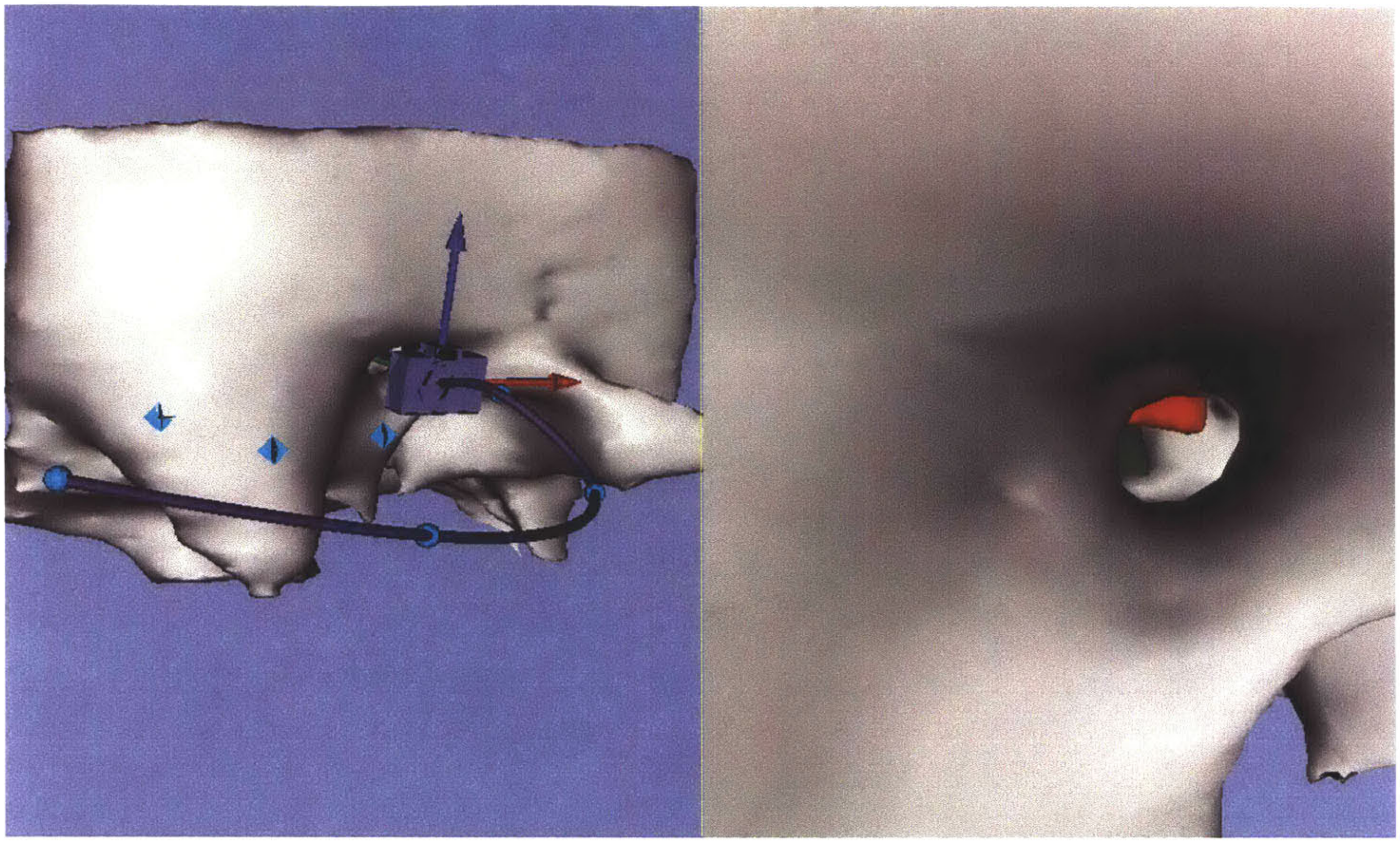


Figure 1-9: A fly-through around the temporal (ear) bone. On the right screen, the point of view of the endoscope looking down the ear canal where a tip of the facial nerve can be seen in red. The trajectory path on the right screen was defined semi-automatically by picking landmarks (blue diamonds) on the surface of the bone.

scanners were reformatted and studied. 3D surface and volume rendered views were generated, and a virtual endoscopy movie was created to compare the images from VCT with those from MSCT. This study demonstrated that the overall image quality was better with VCT than with MSCT. Finer anatomical details such as the the articulations in the ossicular chain, the facial nerve along with all its branches, the osseous spiral lamina in the cochlea, and numerous other structures were visualized with much less partial volume effect. Smaller lesions could be seen in the VCT data as compared with the MSCT data.

For this study, virtual endoscopy was crucial. It would be impossible to fully visualize and compare spatially the structures of the ear in different scans just by looking at the slices because the resolution of the scanner is very high, and the

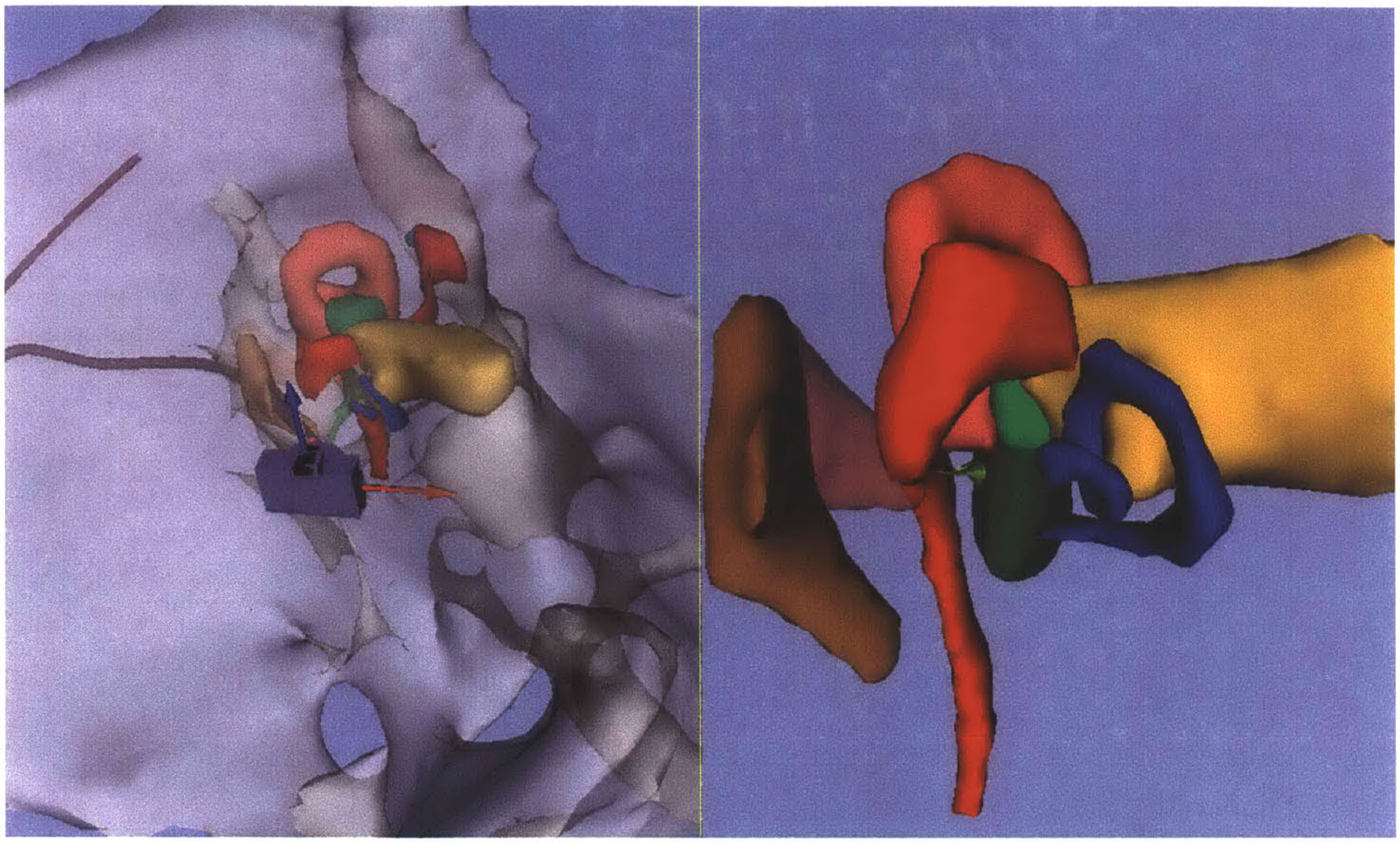


Figure 1-10: A fly-through inside the inner ear. The right screen shows the position and orientation of the endoscope inside the temporal bone. The endoscope is looking at the spatial relationship between the ossicular chain (in brown), the cochlea (in blue) and the facial nerve (in red). The right screen shows the point of view of the endoscope and the temporal bone is made transparent for better visibility.

density of information on the slices is also very high so a slice-by-slice comparison would be too time consuming and not informative. Reconstructing 3D models and exploring them with virtual endoscopy proved to be the right method of comparison.

Some sample views of the temporal bone and middle and inner ear structures are shown in Figures 1-9 and 1-10. Figure 1-9 shows the temporal bone (ear bone), at the entrance of the ear. The user has decided to create a fly-through manually where the endoscope first starts outside the ear to show the structure of the temporal bone and finished inside the ear to show important ear structures such as the ossicular chain (three tiny bones that work together as a lever system to amplify the force of sound vibrations that enter the ear canal), the cochlea (responsible for converting sounds which enter the ear canal, from mechanical vibrations into electrical signals) and the facial nerve structures. It is very valuable to create a fly-through that takes the viewer inside the ear to explore how these structures occupy the ear cavity and how they are positioned with respect to each other. This spatial information is very valuable for the education of physicians and medical students and even engineers who would like to understand the function of inner ear structures and their role in human hearing and balance control. Finding the position of the facial nerve can also become crucial for the planning of cochlear implant surgery.

The transparency of models can be set separately in both screens of our software in order to enhance the global understanding. For example in Figure 1-9, the temporal bone is left at its full opacity for the endoscope to see the ear canal entrance. In Figure 1-10, the temporal bone is made semi-transparent in the left screen to allow positioning of the endoscope inside the ear, and completely transparent in the right screen to visualize all the inner ear structures.

The path was created semi-automatically by placing markers at specific points on the surface of structures. When one of these markers is placed on the surface of a model, the endoscope automatically places itself to look at the marked surface. For example in Figure 1-9, for each diamond marker placed on the temporal bone, a sphere marker was added at the position of the endoscope. A trajectory path is then created by interpolating between different positions (sphere markers) of the camera.

We now present our virtual endoscopy software used for these case studies.

1.5.2 Overview of the VE System

In this section, we summarize the most important system requirements for a Virtual Endoscopy system and explain how we addressed those requirements. Details of the design are presented in the later chapter of this thesis.

Interactive Control

We have investigated many tools to interactively control and move the endoscope and after several iterations of the design, have found that the gyro and the picker tools are very intuitive and simple to use and have received positive feedback from clinical users. The gyro is a tool to intuitively drag and rotate the endoscope and the picker is a mechanism to select landmarks on the surface of models and have the endoscope immediately jump to observe that particular landmark. Intuitive interactive navigation is important for users to manually explore the datasets and discover interesting points of view in minimal time.

Trajectory Creation

Most clinical users would rather use a fly-through than an interactive exploration. Once they have flown through the data, they might be interested in manually exploring some suspicious areas that they detected with the initial fly-through.

We address these needs in our software by providing three modes of trajectory creation: automatic, semi-automatic and manual. As explained previously, the path is used as a 'rail' to guide the virtual endoscope during a 'fly-through'. The path creation can be completely manual and interactive, by moving the virtual endoscope with the gyro and recording the motion of the endoscope. It can be semi-interactive by selecting points of view with the picker and letting the program interpolate the trajectory between those points, as shown in the ear case study. Finally, it can be completely automatic, by picking a start and end point on a model and letting the program find the best trajectory from the start to the end, as shown in the colonoscopy case study. Once a trajectory is created, it can be recorded and saved to a file. When the endoscope is flying through a model by following a trajectory, it can be interrupted at any time and manually moved around to explore the data more closely or at a different position. In addition, the trajectory can be modified interactively.

Multiple paths can be displayed and fly-throughs can happen synchronously. We designed an algorithm that non-linearly matches centerline points based on the geometric information of the organ, described in Chapter 4.

Cross-References

Another important feature for both inner and outer surface viewing is the ability to display the original grayscale volume data on reformatted slices and track particular landmarks in the volume, as shown in the colon case study. The reformatted slices can be oriented orthogonal or parallel to the camera view plane. The center of the slice can be positioned at the camera position, at the focal point position or at the surface point picked by the user. The user has the option to display the reformatted planes in the 3D view, or just look at them in the 2D windows. This cross-reference is highly valued by physicians since often there is information on the grayscale image not fully represented in the 3D surfaces.

1.5.3 Why is Virtual Endoscopy useful?

The aim of virtual endoscopy is not to replace conventional endoscopy but to enhance the procedure and increase its success rate by providing a valuable surgical planning and analysis tool. Virtual endoscopy can also be of great value for medical education and post-surgical monitoring. Clinical studies have shown that virtual endoscopy is useful for surgical planning by generating views that are not observable in actual endoscopic examination and can therefore be used as a complementary screening procedure and as a control examination in the after care of patients [10, 11, 12, 13, 14].

There are several applications or procedures that can benefit from virtual endoscopy:

Surgical Planning

The goal of surgical planning is for the physician to acquire enough information to determine with confidence that a surgery is needed and that it will be beneficial. Conventionally, medical scans and human expert medical knowledge are combined to produce a diagnosis. Any extra source of information can be critical to help in the diagnosis and this is where virtual endoscopy can help.

With virtual endoscopy, the surgeon can quickly and intuitively explore the huge amount of data contained in a medical scan because it is displayed in a compact and intuitive form of 3D surface models. The surgeon can quickly fly-through a patient-specific model without having to operate on the patient or even requiring the presence of the patient during the diagnosis. If a suspicious shape is detected, the physician can cross-reference back to the exact location of that shape in the original volume, displayed on a plane oriented in a meaningful way (for example perpendicular to the organ wall). Furthermore, the surgeon can record a trajectory through the organ for later sessions, save a movie and show it to other experts for advice and save it in a data-bank for other surgeons who will come across similar cases.

Being able to explore the virtual organ in a similar fashion to the actual surgery can be a valuable tool for the physician to mentally prepare for the surgery. It can also be used to train medical students to practice the surgery virtually and become familiar with the anatomy.

Virtual Endoscopy can also provide unique information that is not available directly from conventional endoscopy or medical scans. The first category of information is provided by analysis tools such as measurement, tagging and surface analysis. The user can place markers directly on the 3D surface and measure a distance or a curvature between markers. This gives the physician 3D information about the extent of a detected lesion. The surface of the 3D model can also be directly color-coded to highlight suspicious areas based on curvature or wall thickness information [15], [16], therefore bringing the physicians attention to those areas for further diagnosis. Virtual endoscopy can be very informative to the physician about potential abnormalities that a patient may have and is therefore a very useful surgical planning tool, as described by many clinical studies [10, 11, 12, 13, 14].

Synchronized Fly-throughs

Another advantage of our software is the ability to show synchronized fly-through of aligned datasets of the same organ. As explained in Section 1.3, it is sometimes useful to scan a patient in different body positions in order to obtain images of an organ in different orientations and subject to different gravity and pressure forces to determine whether there are artifacts by comparing images. If a 3D model of the organ is reconstructed from each dataset, then it would be very beneficial to show synchronized fly-throughs inside the models and guarantee that the position of the endoscope in each colon is identical at each step of the fly-through. The physician could then compare views at the same location inside the organ and more easily find artifacts that appear on one image but not the other. In addition, slices can be reformatted to be parallel to the camera plane and saved for each fly-through, similarly to Figure 1-8. This would produce a registered stack of 2D images across all the datasets, and therefore indirectly produces a registration of the volumetric data.

To solve this problem, our method described in Chapter 4 uses dynamic programming and geometric information to find an optimal match between sampled centerpoints. Once the centerlines are matched, fly-throughs are generated for synchronized virtual colonoscopy by stepping incrementally through the matched centerpoints.

This centerline registration algorithm could also be applied to misaligned datasets acquired before and after a patient's treatment. Organs can be non-linearly deformed during a treatment or a surgery or just due to body fluctuations in time. Our algorithm could be useful for cases when a physician wishes to monitor the success of a therapy by comparing a pre-treatment and a post-treatment scans.

Medical Education

The ability to create a trajectory semi automatically by picking points on the surface of the models is very useful for medical students and doctors to explore a set of 3D models easily and quickly. This saves valuable time since novice users can easily be frustrated by 3D navigation. In addition, the trajectory created can be saved to a file, so it can be viewed at later times or shown to medical students and the medical community as a learning tool. The inner ear fly-through described in our third case study is a good example of a educational use of our software.

1.5.4 Limitations

Virtual endoscopy also has some limitations. The main limitation is due to the resolution of the medical imaging scans as well as the segmentation of the structures in those scans. If the resolution of the scans does not permit imaging of certain lesions, then those lesions will not be present in the virtual datasets. In this case, conventional endoscopy remains the only good option for diagnosis. The quality of the segmentation can also be a limiting factor since the quality of the 3D model produced and its resemblance to reality is a direct result of how good the segmentation is. Higher resolution medical scanners and more accurate segmentation are active areas of research and significant progress in these fields will also increase the quality of virtual endoscopies.

However, even with those limitations, clinical studies [10, 11, 12, 13, 14] have demonstrated the benefit of virtual endoscopy.

1.6 Thesis Contributions

This thesis brings three main contributions to the field of surgical planning and virtual endoscopy:

An Intuitive Visualization Platform with Image Fusion -

The first contribution of this thesis is an interactive visualization tool that is intuitive to use and that provides extensive cross-reference capability between the 2D and 3D data. This platform combines many features in one environment: interactive manual navigation and path creation, guided navigation, cross references between the 3D scene and the original grayscale images, surface analysis tools, synchronized fly-through capabilities. This contribution is important since we believe that a better pre-operative visualization leads to better surgical planning.

A Central Path Planning Algorithm -

The second contribution of this thesis is an automatic path generation algorithm integrated in the visualization tool that takes as inputs a 3D model or a labelmap volume and two end points defined by the user, and outputs an optimal centerline path through the model from one endpoint to the other. This path can then be used to move the virtual endoscope through the model for an exploration of the inner surface. Automatic path generation can save surgeons valuable time by providing them with a full trajectory in a few minutes and freeing time for the most important task which is the manual exploration of critical areas identified with the first pass of the fly-through.

A Centerline Registration Algorithm Based On Geometric Information -

We developed a method for the registration of colon centerlines extracted from a prone and a supine colon scans. Our method uses dynamic programming and geometric information to find an optimal match between sampled centerpoints. Once the centerlines are matched, fly-throughs are generated for synchronized virtual colonoscopy by stepping incrementally through the matched centerpoints.

1.7 Thesis Layout

In Chapter 2, we describe our interactive Virtual Endoscopy system. In Chapter 3, we present an automatic path planning algorithm that produces center points used by our system to fly-through anatomical models. In Chapter 4, we present our centerline registration algorithm. We present different applications that used our virtual endoscopy system throughout the chapters and we conclude this thesis in Chapter 5.

Chapter 2

The Interactive Virtual Endoscopy Module

In this chapter, we present in detail the system architecture of our Virtual Endoscopy (VE) software and its integration in the 3D Slicer, an open-source software package for visualization and analysis of medical images. We first introduce computer graphics concepts of virtual cameras and 3D visualization. We then briefly present the 3D Slicer platform and some of its important features relevant for our module. In the rest of the Chapter, we present the VE module and its features in detail, along with example applications of our software. We conclude this section with a discussion of related work.

2.1 Computer Graphics Primer

2.1.1 Scene and Actors

In computer graphics, 3D objects that are rendered on the screen are called *actors*. Actors are placed in a scene, called the *world*, and their global position and orientation is encoded in a *world matrix* associated with each actor. One of the major factors controlling the rendering process is the interaction of light with the actor in the scene. Lights are placed in the scene to illuminate it. A virtual camera is also positionned

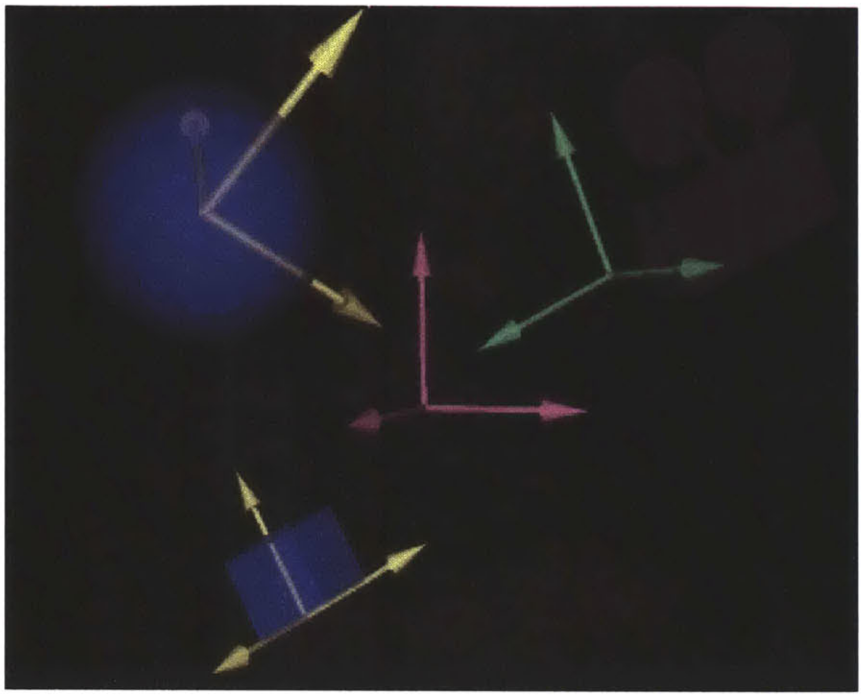


Figure 2-1: all the actors (the golf ball, the cube, the virtual camera) have a local coordinate system shown by their axis. These local coordinate systems can be expressed in terms of the global (world) coordinates shown by the pink axes in the center of the image.

in the world to "film" it, by capturing the emitted light rays from the surface of the actor. A virtual camera does not usually have a graphical representation since it is behind the scene, but it has a global world position and orientation, just like other actors (see Figure 2-1).

2.1.2 Camera Parameters

To render a 3D scene on the 2D computer screen, the light rays emitted from the scene are projected onto a plane (see Figure 2-2). This plane belongs to the *virtual camera*. The position, orientation and focal point of the camera, as well as the method of camera projection and the location of the camera clipping planes, are all important factors that determine how the 3D scene gets projected onto a plane to form a 2D image (see Figure 2-2).

- The vector from the camera to the focal point is called the *view normal* vector.

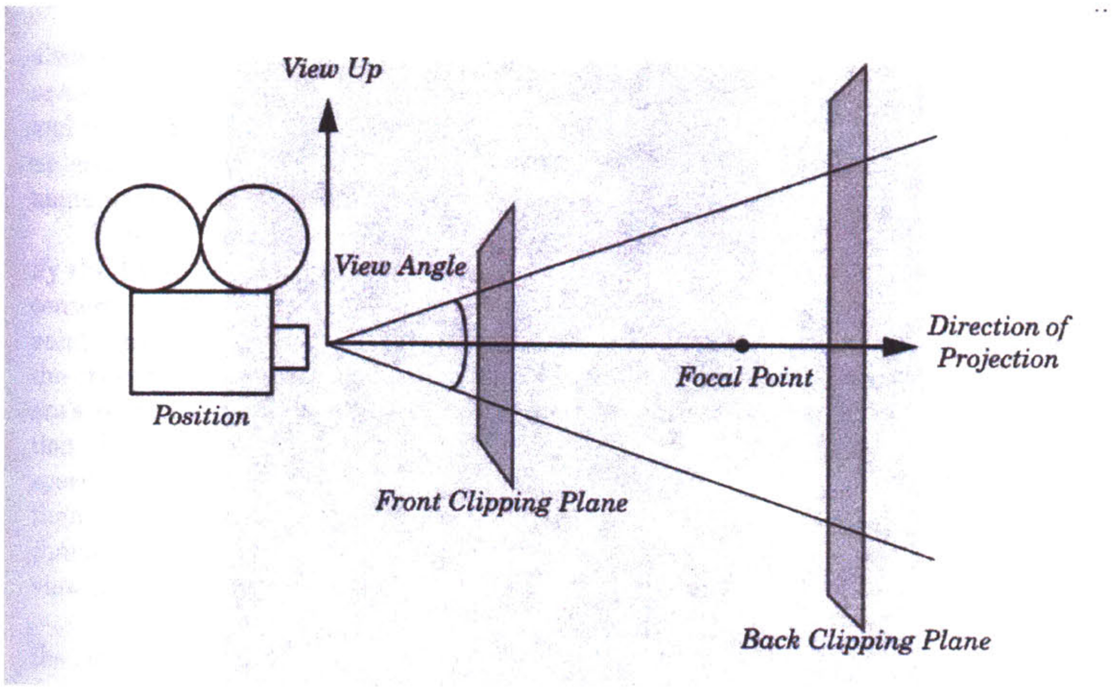


Figure 2-2: Virtual Camera Parameters

The camera image plane is located at the focal point and is perpendicular to the view normal vector. The orientation of the camera and therefore its view is fully defined by its position and the position of the focal point, plus the camera *view up* vector.

- The front and back clipping planes intersect the *view normal* vector and are perpendicular to it. The clipping planes are used to eliminate data either too close or too far to the camera.

- *Perspective projection* occurs when all light rays go through a common point and is the most common projection method. In Figure 2-2 for example, all rays go through a common point. To apply perspective projection, we must specify a camera view angle.

2.1.3 Navigation

In most 3D rendering systems, the user navigates around the scene by interactively updating the calibration parameters of the virtual camera. The user can move or rotate the virtual camera with six degrees of freedom with the help of pre-defined mouse and keyboard actions. An *interactor* translates mouse and keyboard events into modifications to the virtual camera and actors.

2.2 The 3D Slicer

The 3D Slicer is an open-source software package that is the result of an ongoing collaboration between the MIT Artificial Intelligence Lab and the Surgical Planning Lab at Brigham & Women's Hospital, an affiliate of Harvard Medical School. The 3D Slicer provides capabilities for automatic registration (aligning data sets), semi-automatic segmentation (extracting structures such as vessels and tumors from the data), generation of 3D surface models (for viewing the segmented structures), 3D visualization, and quantitative analysis (measuring distances, angles, surface areas, and volumes) of various medical scans. Developers in many academic institutions[1] contribute their code by adding modules to the software. The Virtual Endoscopy module described in this thesis is an example of such a contribution. In this Section, we present a brief overview of the 3D Slicer. For a more detailed description of the 3D Slicer, we refer the reader to [1].

2.2.1 System Environment

The 3D Slicer utilizes the Visualization Toolkit (VTK 3.2) [17] for processing and the Tcl/Tk [18] scripting language for the user interface.

2.2.2 Overview of Features

Volume Visualization and Reformatting — The 3D Slicer reads volume data from a variety of imaging modalities (CT, MRI, SPECT, etc.) and medical scanners. The volume data is displayed in Slicer on cross-sectional slices, as

[1]including MIT, Harvard Medical School, John Hopkins University, Ecole Polytechnique Federale de Lausanne (EPFL), Georgia Tech and many other international institutions.

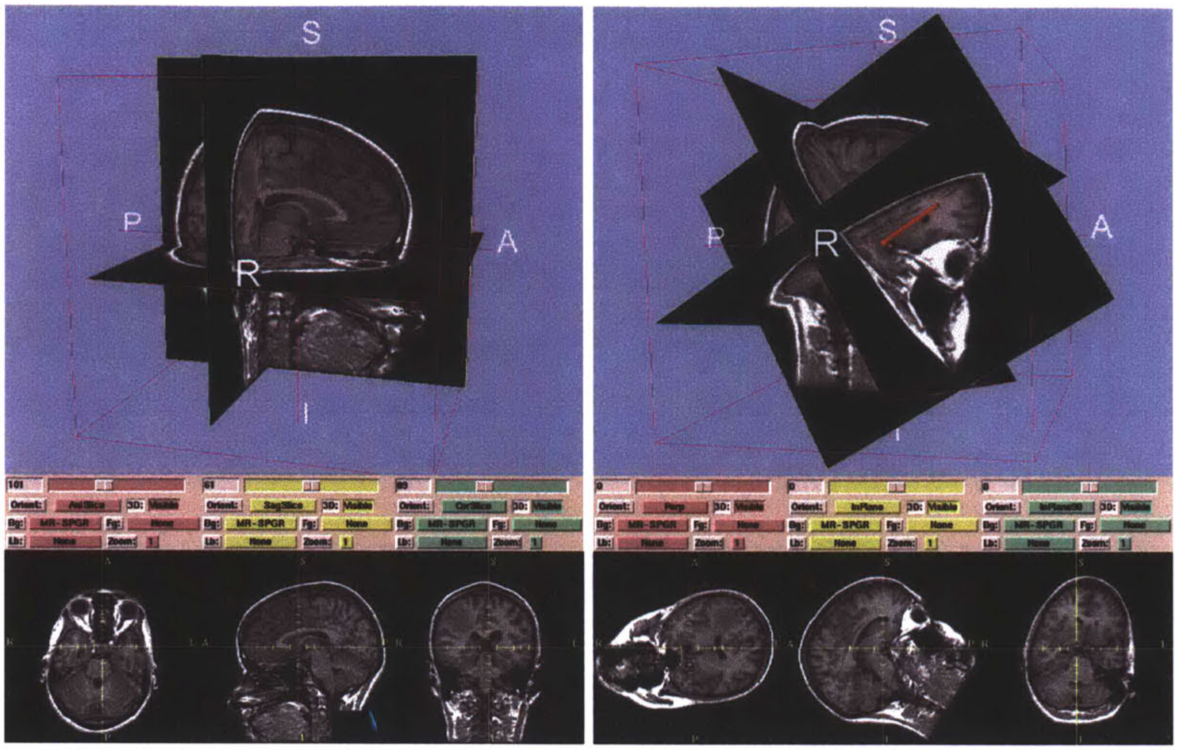


Figure 2-3: Left: slices oriented relative to the reference coordinate frame. Right: slices oriented relative to the pointing device (orange).

shown in Figure 2-3. Each 2D pixel of the slice plane is derived by interpolating the 3D voxels of the volume data intersected by the plane. In the 3D Slicer, up to 3 slices can be arbitrarily reformatted and displayed in the 3D view. Each slice is also displayed in a separate 2D screen (bottom screens of Figure 2-3).

3D Model creation and Visualization — To better visualize the spatial relationships between structures seen in the cross-sectional views, the 3D Slicer provides tools to create 3D models. Since the 3D Slicer is an open-source software, many researchers around the world have integrated their segmentation algorithms in the software. The user therefore has the choice to manually outline structures on the slice or use a semi-automatic or automatic algorithm that will outline particular structures. The output of the segmentation is a labelmap, a volumetric dataset where each voxel is assigned a value according to which structure it belongs to. For example in Figure 1-6, the labelmap of a tumor is shown on three different slices of a patient's head MRI scan.

Once a 3D model is created, its color and transparency can be interactively changed with the user interface.

Analysis Tools —

The 3D Slicer has a number of analysis tools: volume analysis, tensor data, registration between volumes, cutting tools, measurement tools. For VE, the measurements tools are especially useful since the user can measure distances and angles for shape analysis of lesions such as polyps.

MRML File Format —

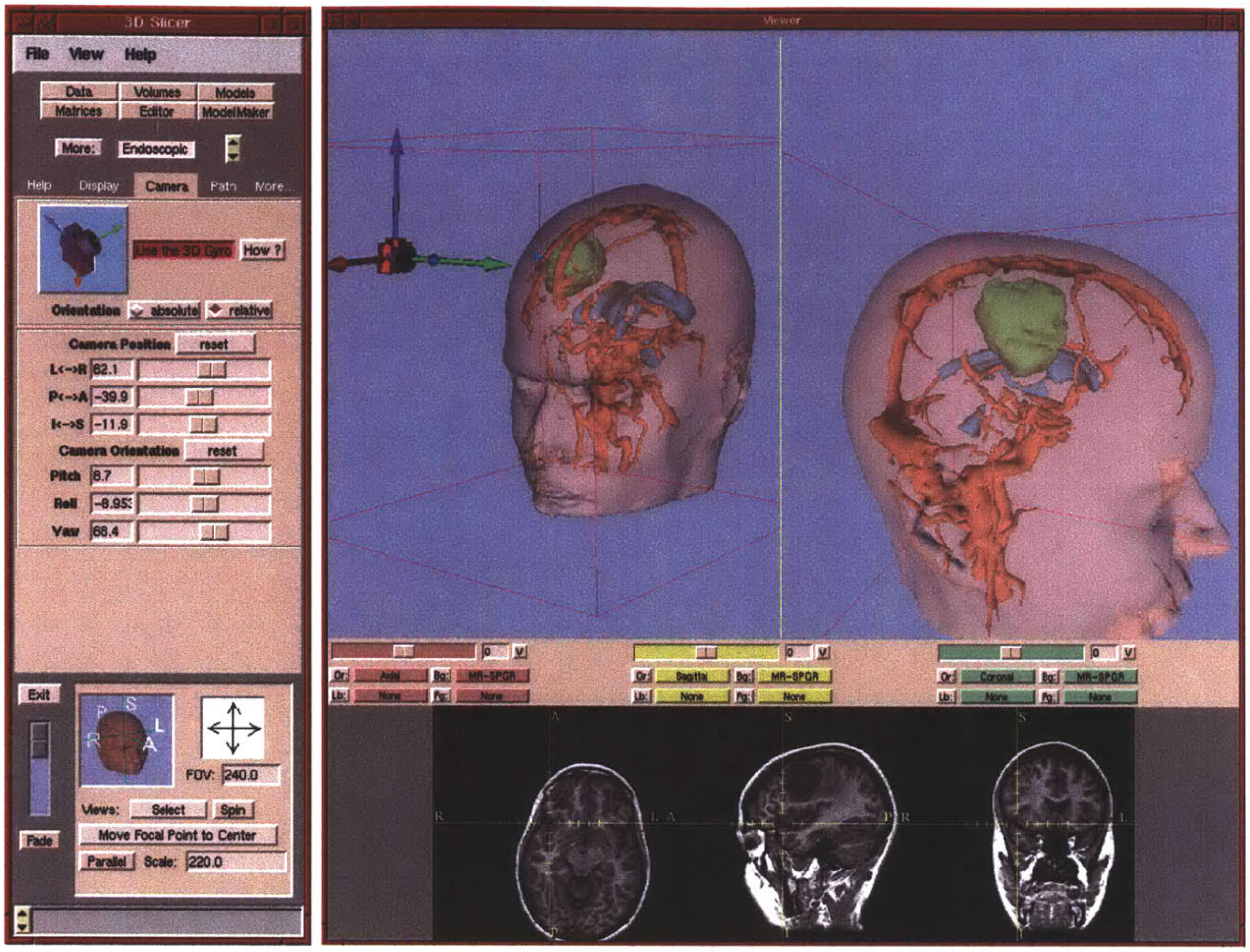
The 3D Slicer uses the Medical Reality Modeling Language (MRML) file format for expressing 3D scenes composed of volumes, surface models and the coordinate transforms between them. A MRML file describes where the data is stored and how it should be positioned and rendered in the 3D Slicer, so that the data can remain in its original format and location. A scene is represented as a tree where volumes, models, and other items are **nodes** with attributes to specify how the data is represented (world matrix, color, rendering parameters etc) and where to find the data on disk. In the MRML tree, all attributes of the parent nodes are applied to the children nodes.

2.3 The Virtual Endoscopy Module

The main contribution of this thesis was the design and implementation of the Virtual Endoscopy (VE) module and its integration in the 3D Slicer. In the rest of this chapter we describe the system requirements and design and implementation challenges for such a program.

2.3.1 Display

A snapshot of the User Interface for the VE module is shown in figure 2-4. On the left, a control panel contains all the commands to interact with the modules. On the



Orientation: absolute [✓ relative]

L<->R 62.1
P<->A -39.9
I<->S -11.9
Pitch 8.7
Roll -8.953
Yaw 68.4
FOV: 240.0
Scale: 220.0

Figure 2-4: The VE program User Interface

right, the user has access to five different screens. The top left screen and the bottom three slice windows are the original screens of the 3D Slicer. The top right screen is the *endoscopic view* described in more detail below.

2.3.2 The Endoscopic View

When the user enters the VE module, a second 3D screen called the *endoscopic view* is displayed along with the original 3D view that we will call the *world view*. In the world view, the user can see an endoscope actor (see the top left screen of Figure 2-4).

The endoscope actor is a hierarchy of three objects:

- the camera that shows the position of the endoscope

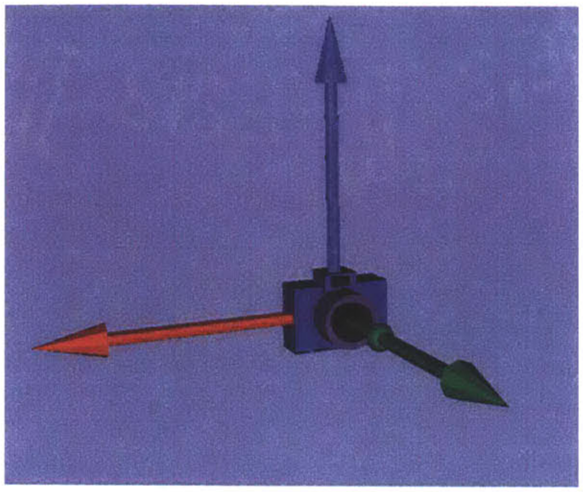


Figure 2-5: The endoscope actor

- the axis, also called *3D gyro*, that show the orientation of the camera (see Figure 2-5).
- the focal point (represented as a green sphere in Figure 2-5) that is a graphical representation of the actual virtual endoscope focal point.

The endoscopic view shows the world from the perspective of the endoscope actor. To obtain this effect, the second screen is in fact produced by a second virtual camera, the *virtual endoscope*, that is positioned and oriented exactly like the endoscope actor. The position and orientation of both the virtual endoscope and the actor endoscope have to be simultaneously updated in order to maintain that effect. We describe how this is achieved in section 2.4.5.

When conducting a virtual endoscopy, it is often useful to change the properties of the actors on one screen but not the other. This allows the user, for example, to track the position of the endoscope in the world view when it is inside a model, by making the model semi-transparent, while keeping the model at its full opacity in the endoscopic view in order to see its internal surface. An example of this is shown in Figure 1-10 showing the exploration of the inner ear in the third case study of Chapter 1. To achieve this effect, when creating another 3D view and adding another virtual camera, one has to also duplicate all the actors that are shown on the world view. In order to facilitate the process of adding another 3D screen to the 3D Slicer

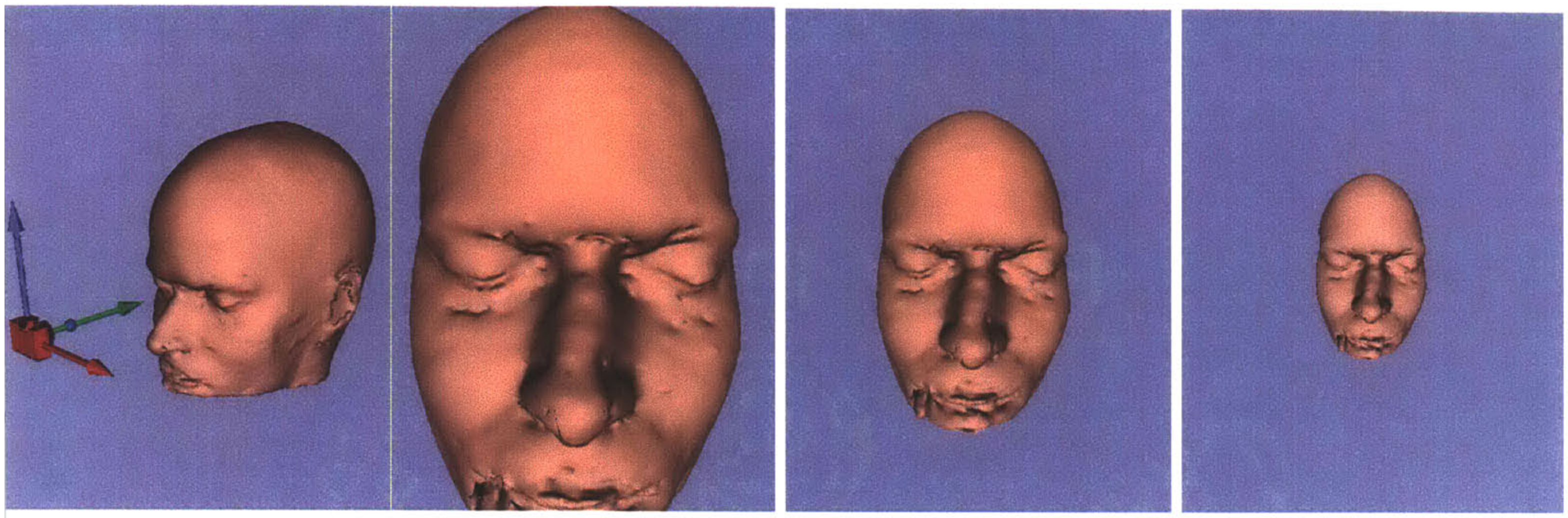


Figure 2-6: On the far left, the world view is shown with the endoscope. The next three screens show the endoscopic view with three different view angles: 120 degrees, 90 degrees, 60 degrees (left to right).

software, we created a general framework where developers can add their own 3D screen. The user interface to interact with the models displayed in that screen is automatically created. Users can then intuitively decide which models are visible and what their opacity is for every screen. This functionality is now used by other modules, such as the Mutual Information Registration module in the 3D Slicer.

To simulate a wide-lens endoscope, the user has the option to change the zoom factor as well as the view angle of the virtual endoscopic camera. A conventional endoscope has a very wide lens (up to 120 degrees). The virtual endoscope therefore defaults to a view angle of 120 degrees, but this value can be interactively changed by the user if she desires to adjust the distortion and the amount of the scene that is rendered. Figure 2-6 shows an example of a human head model seen from three different view angles: 60 degrees, 90 degrees and 120 degrees.

2.4 Navigation

Intuitive 3D navigation is an active topic of research. Novice users can often get disoriented when they have control of a virtual camera filming a 3D scene, so it is important to provide additional information about the position and orientation of the camera. To change the endoscopic view (that is, to control the virtual endoscope) we give the user an option to either move the actor endoscope in the 3D window or to change the view directly by interacting with the endoscopic screen.

Moving and orienting the endoscope should be intuitive so that the user does not spend too much time trying to navigate through the 3D data. We have investigated many ways to control the motion of the endoscope interactively and after receiving feedback from clinical users, we kept the following navigation tools that can be useful in different scenarios.

2.4.1 Interacting directly with the virtual endoscope

One way to move the endoscopic camera is for the user to click and move the mouse over the endoscopic screen to rotate and move the virtual camera. The position and orientation of the endoscope actor is updated simultaneously in the World View as explained in section 2.4.5. This is a great way to move the endoscope to quickly explore a 3D scene and find interesting points of view. It is however not a precise navigation mechanism because we have found that the motion of the mouse gives too many degrees of freedom for translation or rotation of the endoscope. For example, to rotate the endoscope to the right, the user needs to move the mouse in a perfect horizontal line, otherwise the endoscope will also rotate up or down. This is not an easy task and can become frustrating if the user wants to move the endoscope along a specific axis. The 3D Gyro presented in the next section addresses this issue.

2.4.2 Interacting with the actor endoscope

We have found that one of the most intuitive navigation tools is a *3D Gyro.* The gyro is represented in the World View as three orthogonal axes of different colors that are "hooked" to the endoscope as seen in Figure 2-5. The user selects one of the axes by pointing at it with the mouse and then translates or rotates the endoscope along the active highlighted axis by moving the mouse and pressing the appropriate button (left for translation and right for rotation). The 3D Gyro provides six degrees of freedom,

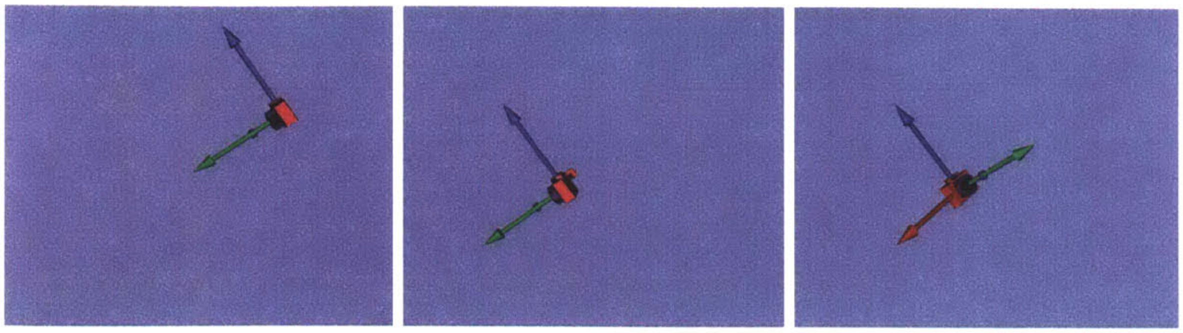


Figure 2-7: This shows translation and rotation in the local coordinate system of the camera: the endoscope is at an initial position on the left screen, on the middle screen, the endoscope has been translated along its x (green) axis and on the right screen, the endoscope is then rotated along its z (blue axis).

but it enables the user to control each degree of freedom separately for a more precise control over the translational or rotational movements of the endoscope[2]. An example of this mechanism is shown in Figure 2-7.

The 3D Gyro tool is intuitive because the user just points at the axis she wishes to select and the axis will translate and rotate along with the motion of the mouse. There is no additional keys or buttons to press. We now detail how we achieved this behavior.

The first step is to detect what the user is selecting with the mouse. To do so, we use a VTK tool called the *picker.* If we detect that the user has selected one of the gyro axes, we override the basic interactor mechanism that is responsible for moving the virtual camera that films the world view and we enter the *endoscopic interaction loop.* The first step of the loop is to detect which mouse button has been pressed to determine if the user wishes to translate or rotate along the selected axis. For this example, let us say that the user chose to translate. We then project the axis selected onto the view plane of the virtual camera by eliminating its depth (z) coordinates. Let V_{axis} be the 2D projection vector of the axis onto the view plane

[2]Before the design of the 3D gyro, we created six sliders for the user to choose exclusively which axes to use for translation or rotation. Each slider controls a degree of freedom (left/right, up/down, forward/backward for translation and rotation). Next to the sliders, the user can also read the world coordinates and angles of the endoscope and change them by hand. However, we realized that the use of sliders can be frustrating since the user has to shift her attention between the screen and the control panel to change the direction of motion. That is why we designed the 3D gyro.

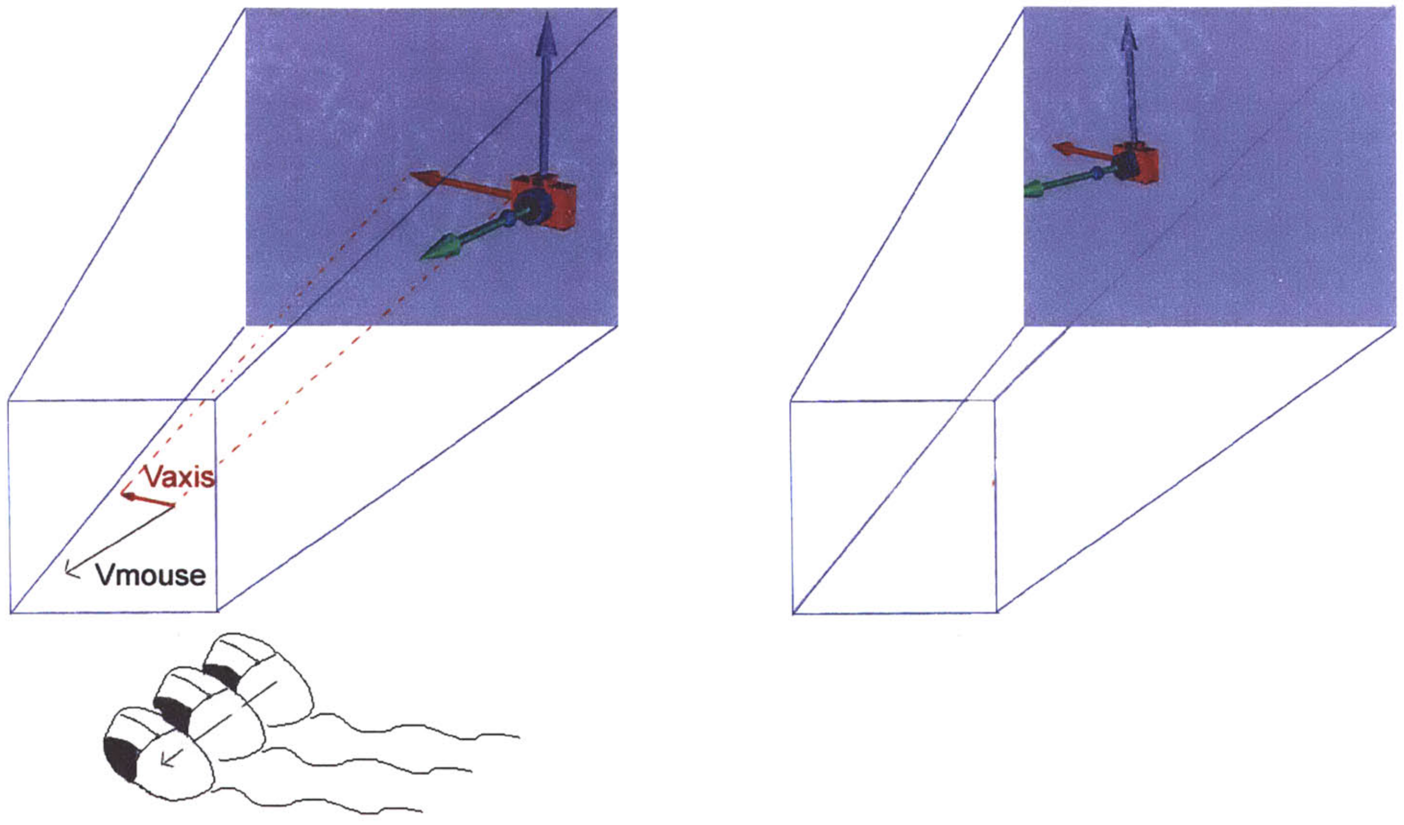


Figure 2-8: This figure shows how the gyro tool is moved. On the left, the user is moving the mouse in the direction of the V_mouse vector and has selected the red axis to translate. The vector V_axis is the projection of the red axis onto the view plane. The angle between V_mouse and V_axis shows the direction of motion along the red axis. In this example, the gyro was translated in the direction of the red axis (right image).

and let V_{mouse} be the 2D vector of motion of the mouse onto the screen, as shown in Figure 2-8. To determine which direction along the axis to translate, we apply the following algorithm:

$$\text{direction of motion} = \left\{ \begin{array}{lll} V_{axis} & \text{if} & V_{axis} \cdot V_{mouse} > 0 \\ -V_{axis} & \text{if} & V_{axis} \cdot V_{mouse} <= 0 \end{array} \right\} \tag{2.1}$$

Intuitively, taking the dot product of the two vectors V_{mouse} and V_{axis} tells us whether the projection of the mouse motion vector on the selected axis is in the *direction* of the axis or in the *opposite direction* of the axis. This tells us precisely along what vector the translation should happen. In the example of Figure 2-8, the angle between the two vectors is less than 90 degrees, so the endoscope is translated along the direction of the selected (red) axis.

To determine how far to move along the axis, we use the magnitude of the mouse motion vector V_{mouse}. The mechanism works identically for rotations. Once we have determined the direction of motion, the type of motion and the magnitude of motion, we update both the actor endoscope and the virtual endoscope, as described in section 2.4.5.

2.4.3 Picking

As an additional feature, the user can pick a point on a model or on a cross-sectional slice (in the World View or in a 2D window) and choose to position the endoscope to look at that world coordinate, as described in the third case study in Chapter 1. This can be useful for a very quick exploration by picking particular points of interest, like in the first case study where we wanted to position the endoscope to look down the right pulmonary vein. This also enables the user to pick suspicious points on a grayscale slice, for example, and to move the camera to that point to explore the 3D model at that position.

2.4.4 Collision Detection

To simulate the surgical environment, we added an option to detect collision between the endoscope and the surface of the models. We check to see if the ray perpendicular to the view plane of the endoscope intersects the surface of a model and detect collision when the distance is under a threshold set by the user. This can be useful when the user manually navigates inside a 3D model and does not wish to go through the boundary walls of the organ. We chose to let collision detection be an option since it can interfere with a free exploration of the 3D dataset by not allowing the endoscope to quickly exit from inside a model.

2.4.5 How this is done

In this section, we explain how the position and orientation of the actor endoscope and virtual endoscope are updated based on the user input.

Matrix Definition

The endoscope has a world matrix called E that defines its position and orientation in world coordinates:

$$E = \begin{bmatrix} viewNormal_x & viewUp_x & viewPlane_x & p_x \\ viewNormal_y & viewUp_y & viewPlane_y & p_y \\ viewNormal_z & viewUp_z & viewPlane_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

The upper 3x3 sub-matrix of E is orthonormal and defines the 3 unit axes of the camera in world coordinates. That sub-matrix therefore defines the orientation of the endoscope. The rightmost column defines the position of the endoscope.

To fully determine the parameters of the virtual endoscope, the focal point position also needs to be specified. If the world matrix and focal point are updated for one representation of the endoscope (virtual or actor), then the other representation also has to be updated to match the change.

Updating the World Matrix

When the user moves the endoscope (virtual or actor), the action sends a command to rotate or translate along a selected axis of the endoscope. To apply those transformations in the local coordinate system of the endoscope, we compute the transformation matrix T that is either the standard rotation matrix or translation matrix used in computer graphics. For example,to rotate around the x axis by θ

$$t_F = \begin{bmatrix} 1 & 0 & 0 & \mathbf{d} \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

(See [19] for more details).

The new E' matrix can be then computed by the matrix operation:

$$E' = T \cdot E \tag{2.4}$$

Updating the Focal Point

Once the position and orientation of the endoscope is updated, the focal point has to be updated as well. Since the focal point is located on the $view\hat{Normal}$ vector, at a distance **d** from the endoscope, its matrix is computed as:

$$F' = t_F \cdot E' \tag{2.5}$$

where

$$t_F = \begin{bmatrix} 1 & 0 & 0 & \mathbf{d} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

2.5 Virtual Fly-Through

The previous section details how to explore the 3D data manually by navigating the endoscope. When the user conducts this manual exploration and finds interesting points of view, it is valuable to save the exploration path, or *trajectory path* through the 3D data. The trajectory path can then be used as a rail to move the endoscope in a "fly-through" and therefore reproduce the exploration automatically. Trajectory paths can be saved in file or recorded as a movie for the physician to share his exploration of the 3D dataset with other medical experts or medical students.

Trajectory paths can also be created **automatically** inside 3D models. This feature saves valuable time for the physician if he wished to use the software to fly inside a hollow organ and mimic a conventional endoscopy to find abnormalities on the organ wall, as described in the second case study of Chapter 1.

In this section, we describe in more detail the different options for the creation and update of trajectory paths for virtual fly-throughs.

2.5.1 The Trajectory Path

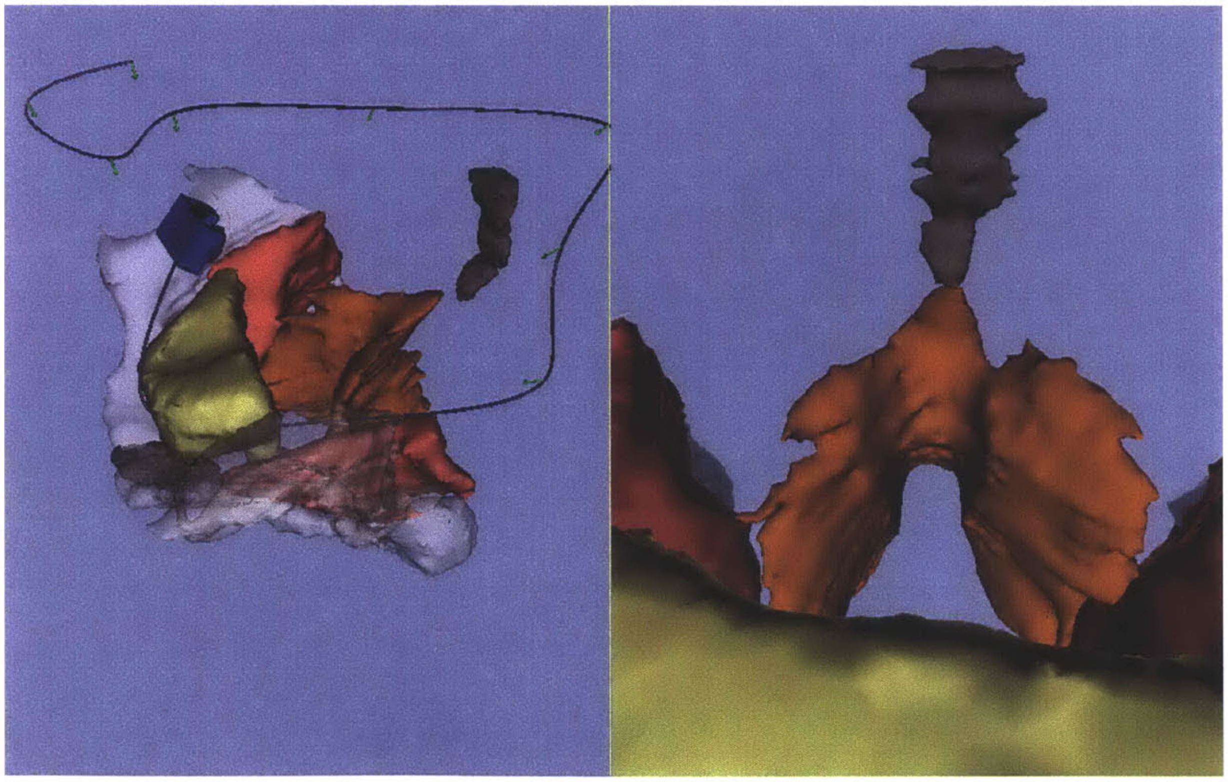


Figure 2-9: A trajectory path exploring organs in the pelvic area. The trajectory path was created manually by moving the endoscope with the gyro and placing landmarks at key areas. The green vector at each landmark records the direction of view of the endoscope. The path is then interpolated between the landmarks.

A fly-through trajectory path is an interpolation between key points called *Land-*

marks.

A landmark has two components:

1. a position in world coordinates represented by a sphere (see Figure 2-10).
2. a direction in world coordinates represented by a vector that shows how the endoscope will be oriented at that location of the fly-through (see Figure 2-10).

As the user creates landmarks, the program interpolates between them using a cardinal cubic spline curve. We chose this interpolation function because it has second-order continuity, i.e the rate of change of the tangent vectors is matched at the landmarks. This ensures smooth transitions in the motion of the endoscope [19]. We parameterize the number of points between two landmarks based on the Euclidean distance between the landmarks so that the camera travels at a relatively constant speed between each pair of landmarks. The path is represented as a tube with 3D vectors at interval points (Figure 2-10).

2.5.2 Landmark Operations

So far, we have explained in detail the different options for controlling the endoscope manually to explore scenes. We now explain the different options to create landmarks in the scene in order to produce trajectory paths.

There are three ways to add a landmark:

1. Interactively
2. Automatically with the Path Planning algorithm
3. From a file

Interactively To interactively record a trajectory, one option is for the user to position and orient the endoscope during a manual fly-through and periodically record a key frame. This creates a landmark at the current position and orientation of the endoscope. The trajectory path is created interactively and updated whenever a new landmarks is recorded.

Figure 2-9 shows an example path created manually to explore pelvic organs. The path first explores the outside surfaces and the spatial relationships of the organs and then enters the bladder where it explores the inner surface.

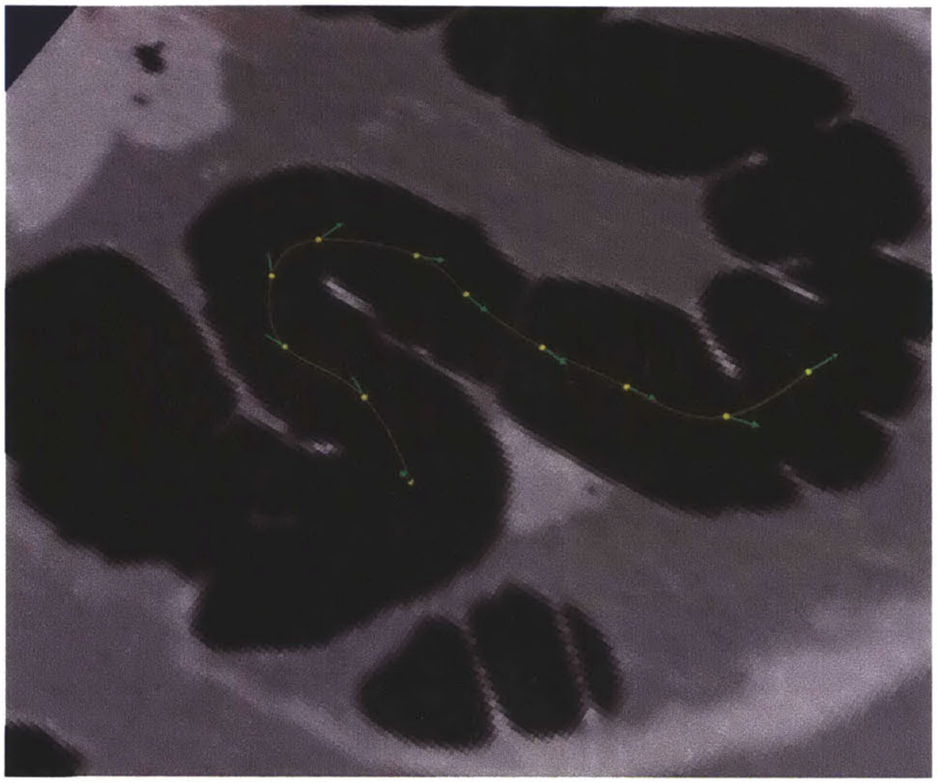


Figure 2-10: A fly-through path defined semi-automatically by placing landmarks on an CT slice of a colon
.

Figure 2-10 shows an example path created semi-automatically by picking markers on a reformatted CT slice that displays a colon along it long axis. The markers were picked on the boundary of the organ wall (the black/grey boundary) as the direction of the vectors show. This can be a quick way to create a path inside an organ, but not the best way since it is not guaranteed that the slice stays in the center of the colon. For this purpose, extracting a trajectory path automatically is much more fast and accurate.

Automatically Automatically extracting a trajectory path inside models of any topology is one of the most popular features of the VE module. To automatically

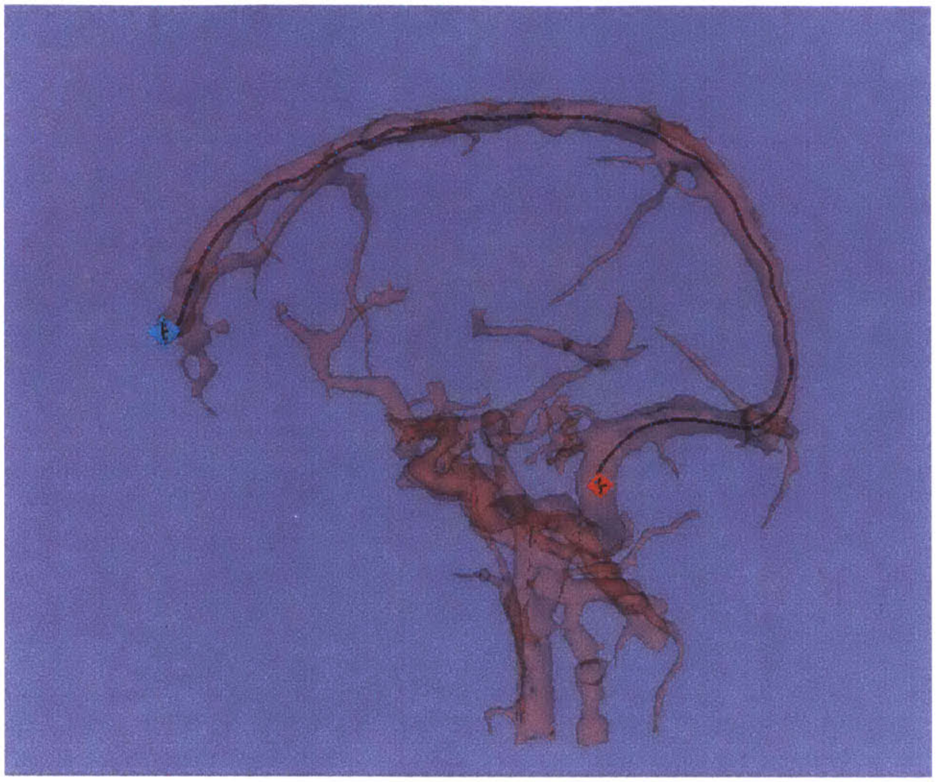


Figure 2-11: A fly-through path inside a brain vessel extracted automatically with the CPP algorithm between 2 user input landmarks (blue and red diamond).

create a trajectory path inside a model, the user has to place a landmark at the entrance of the model and another landmark at the exit and a path that connects both landmarks will be automatically extracted by the CPP (Central Path Planning) algorithm, such that the path is as centered as possible (as far from the organ walls as possible). For example, Figure 2-11 shows a centerline path inside vessels. We detail the CPP algorithm in detail in Chapter 3.

The CPP algorithm outputs a list of centerpoints inside the model. To create a trajectory path, we create a landmark at each centerpoint as follows:

- the position of the endoscope is the position of the centerpoint
- the position of the focal point is the next interpolated landmark on the path. As a result the viewNormal of the endoscope is tangent to the path so the endoscope rests on the path and looks "ahead" during the fly-through (much like a roller coaster wagon).

As each landmark is created, the path is created by interpolating points between the landmarks.

From a File Trajectory paths that are created either manually or automatically can be saved in the MRML file format. Similarly, landmarks coordinates can be extracted with other software programs and written into a MRML file to be loaded and visualized with our software. For these operations, we added two nodes to the MRML API:

```
Path (
   id         int                        // MRML id of this node
   name       string (optional)          // name given to the path
   color      3 floats (optional)        // color given to the path
   rasMatrix 16 floats (optional)        // matrix that transforms the path
)


Landmark (
   id        int                         // MRML id of this node
   position 3 floats                     // 3D position of the landmark
   fp position 3 floats (optional)       // 3D position of the focal point of the
                                         //    landmark
   name      string (optional)           // name given to the landmark
)
```

A fly-through trajectory is represented by a Path node and its children Landmark nodes. A Path node has description attributes (color, name) as well as a transform attribute (rasMatrix). This transform can be used if the path belongs to a particular 3D model and that model is transformed in space. The transform can also be applied to the Path and all its children (therefore the position for each landmark is transformed).

Each landmark needs to have at least a position. Its direction of view is defined by the focal point position (the view normal is the vector from the landmark position to the focal point position). If no focal point position is defined, the default focal point position is the coordinates of the next landmark on the path [4].

2.6 Trajectory Path Update

The user can select any vector or landmark by clicking on it, at which point the endoscope is positioned and oriented at that landmark. The endoscope can then be re-positioned and re-oriented which updates that particular landmark. The user can also delete any landmark. The path is updated and re-rendered in real time after any landmark or vector operation for the user to visualize the result.

2.6.1 Fly-Through

Once the user is satisfied with the trajectory, she can start a fly-through of the model. A fly-through consists of a series of consecutive frames where the endoscope moves through each consecutive interpolated point on the spline. The user can play all the frames at a chosen speed, step through each frame manually, or jump to any frame number.

Figure 2-12 shows selected frames with suspicious shapes that could be polyps during the fly-through of a colon. The trajectory path inside the colon was extracted automatically with the CPP algorithm.

To fully determine the orientation of the endoscope during the fly-through, the View Up vector is interpolated at the time of the fly-through depending on the initial View Up of the endoscope right before the fly-through. The interpolation is as follows: at each landmark, find the plane perpendicular to the viewNormal vector defined at that landmark and project the ViewUp of the previous landmark on that plane in order to obtain the new ViewUp vector. This interpolation ensures a smooth transition of the endoscope, so it never "flips" suddenly from one side of the path to another.

[4]the last landmark is assigned a focal point a few units away in the last direction of view

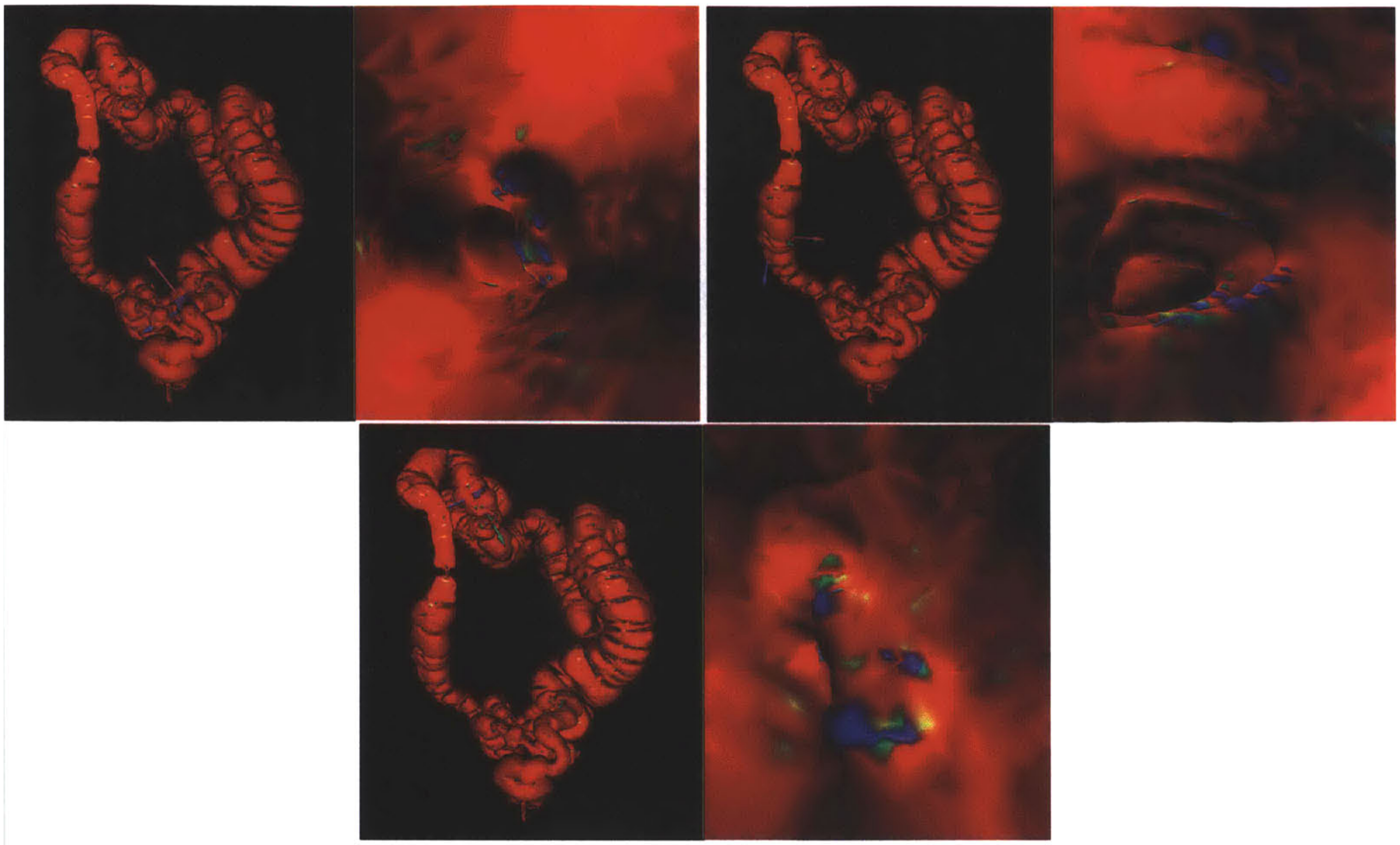
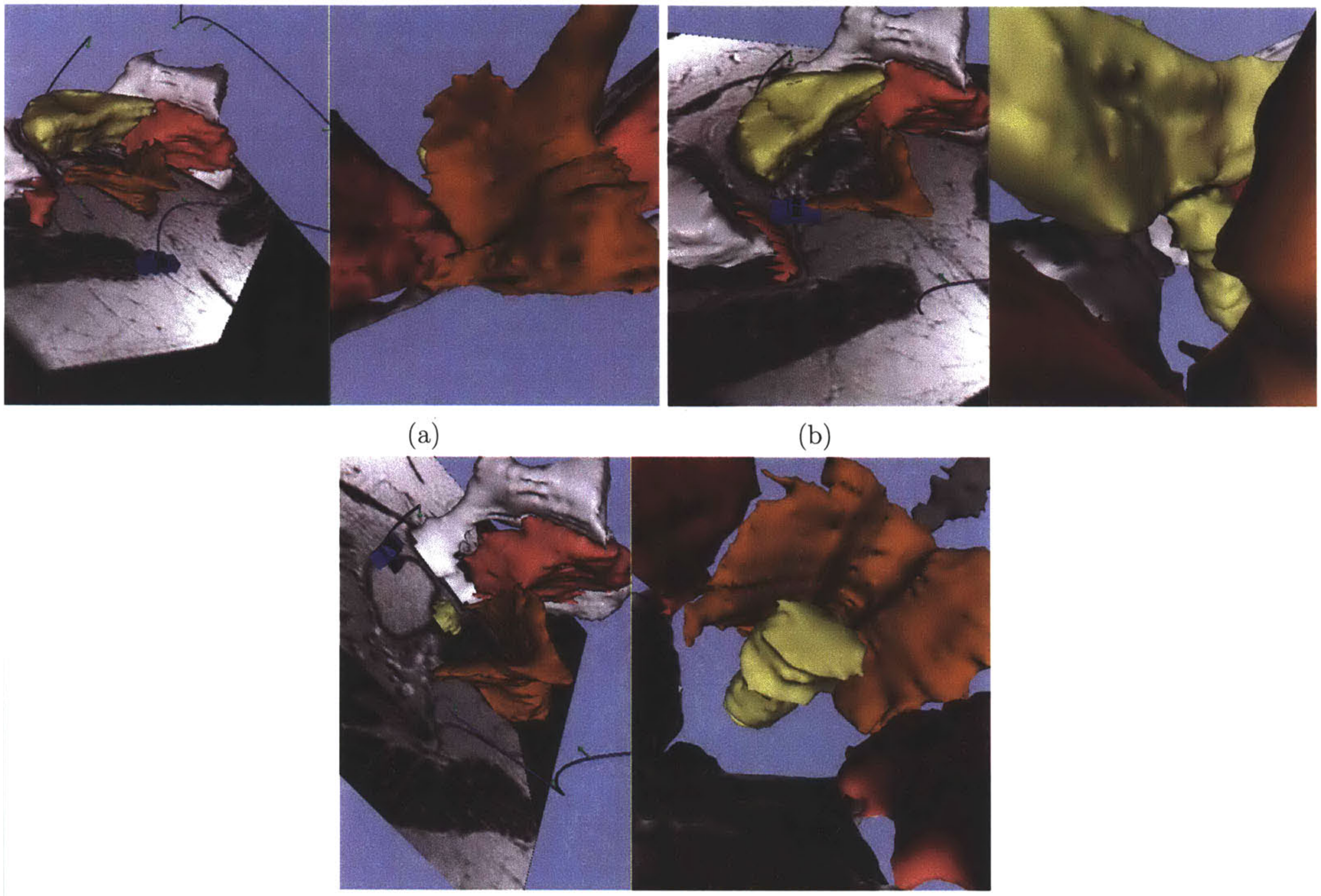


Figure 2-12: selected frames with suspicious shapes that could be polyps during the fly-through of a colon. The trajectory path inside the colon was extracted automatically with the CPP algorithm

2.7 Reformatting Slices for VE

We adapted the plane reformatting mechanism of the 3D Slicer to add an additional visualization advantage to the VE software. The user has the option to reformat the slices in real-time according to the current orientation of the endoscope's axes.

Figure 2-13 shows an example of that mechanism. The endoscope is flying-through pelvic organs to observe the shape of the levator (orange), an organ with important functions that supports the weight of the lower abdomen organs. Studies have shown that the shape of the levator could be used in the diagnosis of prolapse and stress incontinence [20]. Images (a)-(c) show three frames during the fly-through. The grayscale plane, also know as the *reformatted slice*, is kept orthonormal to the endo-



(a) (b) (c)

Figure 2-13: A fly-through of the organs around the pelvic area. The slice shown here is reformatted along with the endoscope orientation.

scope's view and centered at the endoscope position. The reformatted slice brings additional information to the physician about other anatomical structures not reconstructed in 3D.

Choosing the center of the Reformatted Slice The center of the slices, that is the position where the 3 orthogonal planes intersect, can be determined by the user. For example in Figure 2-14, the endoscope is looking at a bladder that has been colored based on wall thickness, (a tumor appears in blue center of the endoscopic

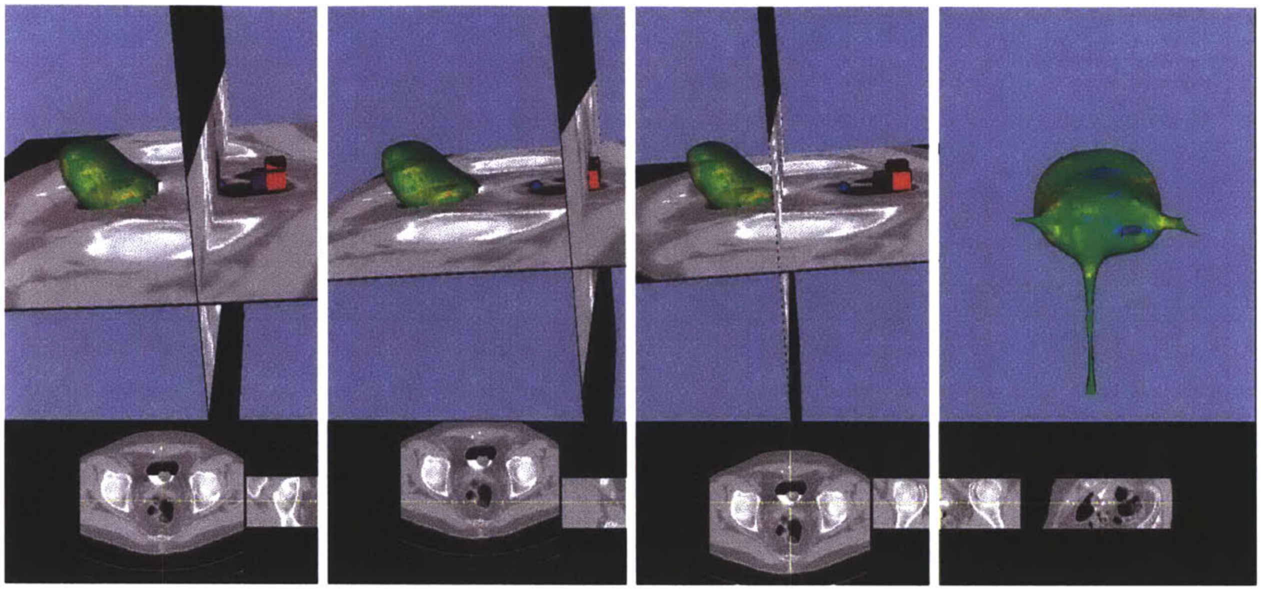


Figure 2-14: The figure shows a virtual cystoscopy where 2 slices reformatted in the coordinate system of the endoscope and positionned at: the focal point (left image), the camera (second image from the left), the surface point (third image from the left). The Endoscopic view is shown on the far right image.

screen) and 2 of the orthogonal slices are displayed. Each image in Figure 2-14 shows an one of the option for the slice center:

- the position of the focal point (left image in Figure 2-14),
- the position of the endoscope (middle image in Figure 2-14),
- a surface point picked by the user on any 3D models (right image in Figure 2-14).

Choosing the position of the Reformatted Slice The user can select to reformat the slice in any of the three planes defined by its local coordinate axis. Choosing which plane is most informative highly depends on the organ being explored and the type of information desired from the grayscale image.

In the case of Figure 2-13, the reformatted slice that is perpendicular to the view plane of the endoscope and to its blue axis is particularly informative about the organs being viewed.

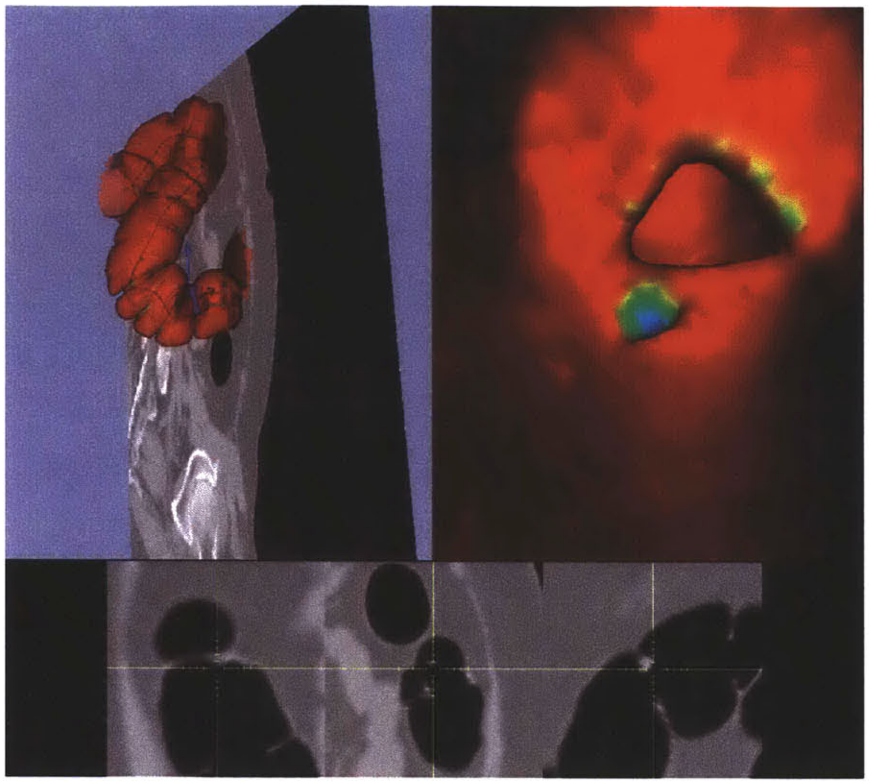


Figure 2-15: During a colonoscopy, the three reformatted slices shown at the bottom are oriented in the coordinates of the endoscope and centered at the blue polyp shown on the endoscopic view (top right). The polyp seen in the endoscopic view can also be detected on the grayscale images.

Another example is shown in Figure 2-15, where the slice displayed in the World View is centered at the polyp shown in blue in the Endoscopic View and oriented parallel to the lens of the endoscope (parallel to the red axis of the endoscope). In this case, the plane shown is particularly informative since it shows on one image all the information about the cross-section of the colon, the size of the polyp in the cross-sectional plane as well as its exact position in the body.

All three reformatted slices can also be shown in the slice windows (bottom three windows of Figure 2-15. The middle slice is the one shown in 3D on the upper left screen. The other two slices are not shown in 3D, otherwise they would eclipse the 3D model. The left slice is the one that is orthogonal to the view plane and parallel to the endoscope's green axis. The right slice is the other orthogonal slice, parallel to the blue axis of the endoscope. All 3 slices are centered at the the blue polyp that can be seen on the endoscopic screen (top right screen). This cross reference back to the grayscale data visualized in a meaningful way is very valuable to physicians to check the information they see on the 3D models. Here, the grayscale information confirms the extent of the polyp in the three reformatted places.

2.8 Multiple Paths and Synchronized Fly-Throughs

Multiple Paths can be created and visualized at once. We also created a mechanism where synchronized fly-throughs of two different paths can happen at once. We detail this mechanism and its motivation in more detail in Chapter 4.

2.9 Saving Data

At any time, the user can save a data file with the positions of the existing landmarks as well as the model properties in a MRML file [1]. When a user-created data file is loaded, our system automatically creates a path from any landmarks defined in the file as explained in Section 2.5.2. This allows the user to calculate landmark positions with other programs if they wish and use our program to visualize them along with the trajectory that they define. The user can also save all frames during a fly-through to make a movie.

2.10 Related Work

The Virtual Endoscopy Center at Wake Forest University School of Medicine has created a Virtual Endoscopy System called FreeFlight [21] that explores 3D surface models. The surface models are created by automated image segmentation specific to each organ based on thresholding and region growing. In FreeFlight, the user navigates inside the surface model by choosing the motion of the camera with a mouse click (forward, no movement, backward) and rotates the camera up and down or left and right with the mouse motion. We have found that a user rarely moves the mouse in a perfect vertical or horizontal motion, so it is hard to only rotate up or rotate only to the right with such commands. Our 3D Gyro described in Section 3 allows the user to navigate with a single degree of freedom at a time for a more precise motion. In FreeFlight the user moves the camera in reference to what she sees on the endoscopic view. We have found that in order to navigate in a full 3D scene, it is also useful to have an actor representing the endoscopic camera in the scene with the anatomy and to have an intuitive way to move this actor with six degrees of freedom along its own coordinates or along the absolute coordinates of the 3D scene.

The Multidimensional Image Processing Lab at Penn State University [22] and The Visualization Laboratory at the State University of New York at Stony Brook [23] have developed virtual endoscopy programs (Quicksee) in volumetric environments with fast rendering rates. These programs have both a manual mode where the user can explore the scene interactively and an automatic mode where the path is predefined. In interactive mode, the user interface is similar to the one of the FreeFlight program and therefore has the same limitations.

These programs do not allow the user to interactively update the path once it has been created. FreeFlight does not have an option to display a reformatted slice of the volume in the orientation of the endoscope, a feature that our users found very useful. The other two programs use volume rendering, so the user can volume information beyond the surface. However, volume rendering is a time-consuming task and offers slower rendering rates than surface rendering. As fast volume rendering becomes a feature of the 3D Slicer, our virtual navigation program will also be able to explore volumetric scenes. For now the 3D Slicer uses mainly surface rendering in order to run on multiple platforms on computers with non-specialized hardware.

The Virtual Endoscopy Software Application (VESA) developed at the GE Corporate Research and Development Center is a prototype that has been developed in collaboration with the Surgical Planning Laboratory and used for many clinical studies [5, 24, 25, 16]. Many of its concepts have been used in the development of our system.

Chapter 3

Path Planning

In most cases, users will want to use our software to conduct a fly-through inside a particular virtual patient-specific organ, like the colon or esophagus. This fly-through will give them a general idea of the shape of the organ walls for that patient and once they detect an abnormal shape, such as an ulcer crack or a protruding bump like a polyp, they can stop the fly-through and navigate the endoscope closer to the suspicious area to conduct measurements and overlay the grayscale information at that location to see the extent of the lesion beyond the wall and its relation to other surrounding organs.

To create a fly-through inside an organ, generating the path manually can be very time consuming and frustrating for a novice user. For this reason, we provide the surgeon with an automatic path generation button. The scenario would be that the surgeon loads a 3D model, defines a start and an end point and our program returns an optimal path centered inside the model that starts and ends where defined. The user then has the option to fly-through the path and/or edit it manually with the tools described in the previous chapter. This should work for any model, whether the model has branches or not. In Section 3.1, we review previous automatic path generation for virtual endoscopy. In Section 3.2 we describe our Central Path Planning (CPP) algorithm and test it on models of varying topology. In Sections 3.3 and 3.4, we present some optimizations and important parameters of CPP. Finally in Section 3.5, we evaluate the CPP algorithm on datasets of different topology.

3.1 Previous Work

Automatic Path planning for virtual endoscopy is an active field of research where the goal is to automatically extract a fly-through trajectory for the endoscope that stays as far as possible from the organ walls in order to maximize the amount of information that the user sees during the fly-through.

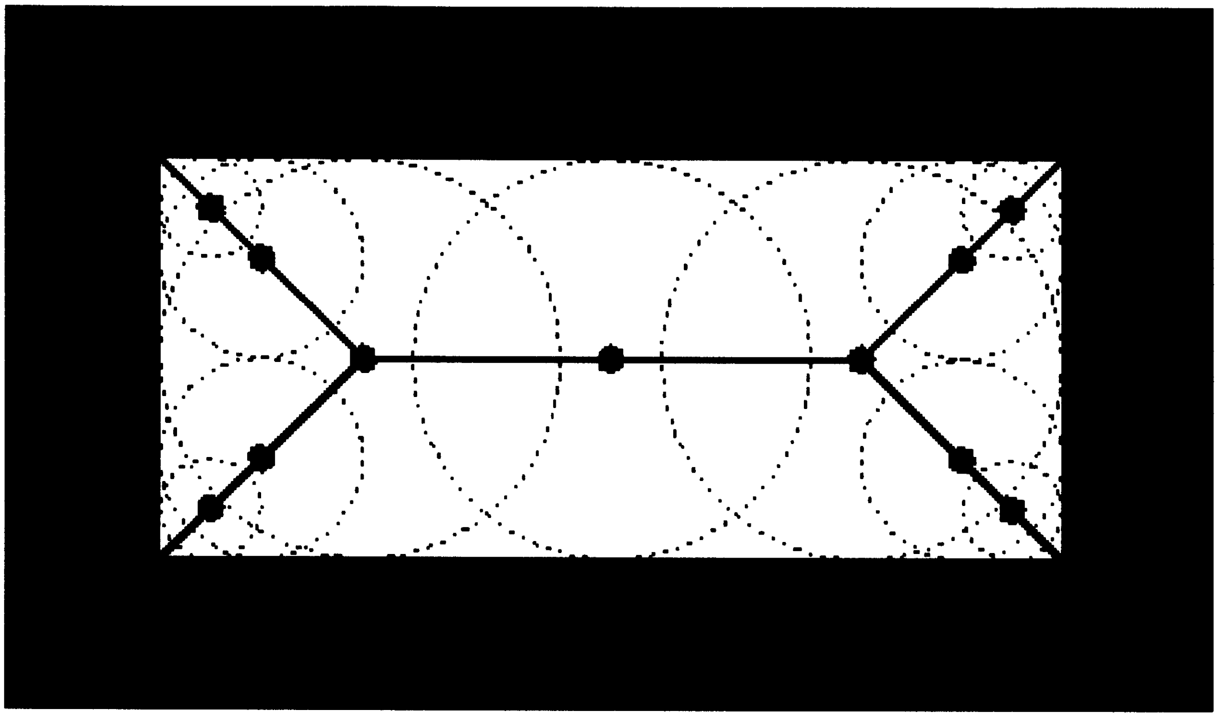


Figure 3-1: Skeleton of a 2D rectangle defined in terms of the center locus of bi-tangent circles

One approach is to automatically extract the medial axis from an object, also called its *skeleton*. A concise definition of the skeleton was given by Blum in [26]: Let us assume that the object is homogeneously covered with dry grass and then set its boundary on fire. The skeleton is defined as the locus where the fronts meet and quench. Another graphical definition of the skeleton is given by the center locus of multi-tangent circles (in 2D) or balls (in 3D) that can be inserted within the shape of the object, as shown in Figure 3-1.

Topological thinning algorithms find a skeleton by peeling off a volumetric object (a labelmap) layer by layer until there is only one layer left, such as the algorithm developped by Kaleem Siddiqi *et. al.* [27]. Depending on the topology of the object, some voxels can form a 2D manifold, which is not an acceptable result for the centerline. Voxels on a 2D manifold are further thinned to produce a one-voxel wide line, adding computational complexity to the algorithm. Once the skeleton is found, it usually consists of one-voxel connected branching paths. From there, the centerline of an object can be defined as the longest path from one end of the skeleton to the other. We adapted a thinning algorithm in the 3D Slicer, but it did not work at interactive speeds on large datasets of more than 10,000K voxels and furthermore did not produce good results for models of certain topologies, such as the colon. One reason is that the colon wall has a lot of symmetric folds and so the centerline extracted have a lot of little branches around the folds. Topological thinning is an excellent method to reduce the complexity of a model by extracting its skeleton, but it is not the most efficient method to extract an optimal trajectory path for virtual endoscopy fly-throughs since the skeleton has to be further processed to produce an acceptable path.

For the colon topology, the center of mass algorithm [28] developped by Steven Haker *et. al.* is probably the most accurate method because it works in the continuous 3D space. It does not attempt to find a global skeleton for the tube-shaped colon, instead it divides the tube in 2D slices and takes the center of mass of each slice to find a centerpoint. The trajectory path is then computed by connecting the centerpoints in order. This method produces high quality paths, but it only works for tube or sphere-like topologies and does not work at interactive speed. To explain this method further, the method for centerline extraction is based on extracting level sets from one end of the colon to the other by using the eikonal equation [29]. Intuitively, both ends of the colon are held at a constant temperature of 0 and 1 degrees respectively and the temperature is free to propagate along the colon. At steady-state, the distribution of temperature u across the surface varies smoothly between 0 and 1 degrees from end to end, and will be free of local maxima and minima away from the boundary curves. Level sets of identical temperatures consist of a loop around the colon surface. The centerline is then formed by the centers of mass of these loops. The method used to find the temperature distribution function is numerical based on finite element techniques. With this method, the average time to extract a centerline from a colon dataset similar to the one used for our evaluation is about 5 minutes which is not as fast as our method for comparable results. Furthermore, the current algorithm cannot be applied to branching structures.

Distance mapping algorithms are another method to extract a centerline automatically and at interactive speed.

Olivier Cuisenaire at the Universite Catholique de Louvain [30] has designed an algorithm for centerline extraction inside a 3D model. His algorithm first uses a Distance Transform to generate a shortest path from one endpoint of the model to the other. The problem with this approach is that the path "hugs" the walls in order to be as short as possible. He then uses a Snake energy minimization algorithm to re-center the path. We have identified that at certain sharp turns, the path is still not centered perfectly. Thomas Deschamps [31] has developped another algorithm by using the Descartes-Fermat principle of light propagation inside an object based on the grayscale values of the image and then penalizing the speed of propagation inside the object by making the propagation faster in areas further from the walls of the object. This method avoids the initialization step of the snake algorithm and problems of local minimas. The first step of our approach is similar to Deschamp's since we apply a distance transform algorithm directly from the walls of the object. Since the distance transform naturally tells us which voxels are the most centered, we then use Dijkstra's algorithm to find a unique path between two points chosen by the user, even for a branching topology. This allows us to keep the algorithm simple and efficient and give more power to the user by allowing them to specifically chose the endpoints of the path that are not necessarily the endpoints of the model.

3.2 Centerline Path Planning (CPP) Algorithm

Conceptually, our algorithm first finds the most centered voxels in a labelmap using an Euclidean Distance Transform from the Boundary (EDFB) and then finds the most centered path between two user-selected points using an optimized version of

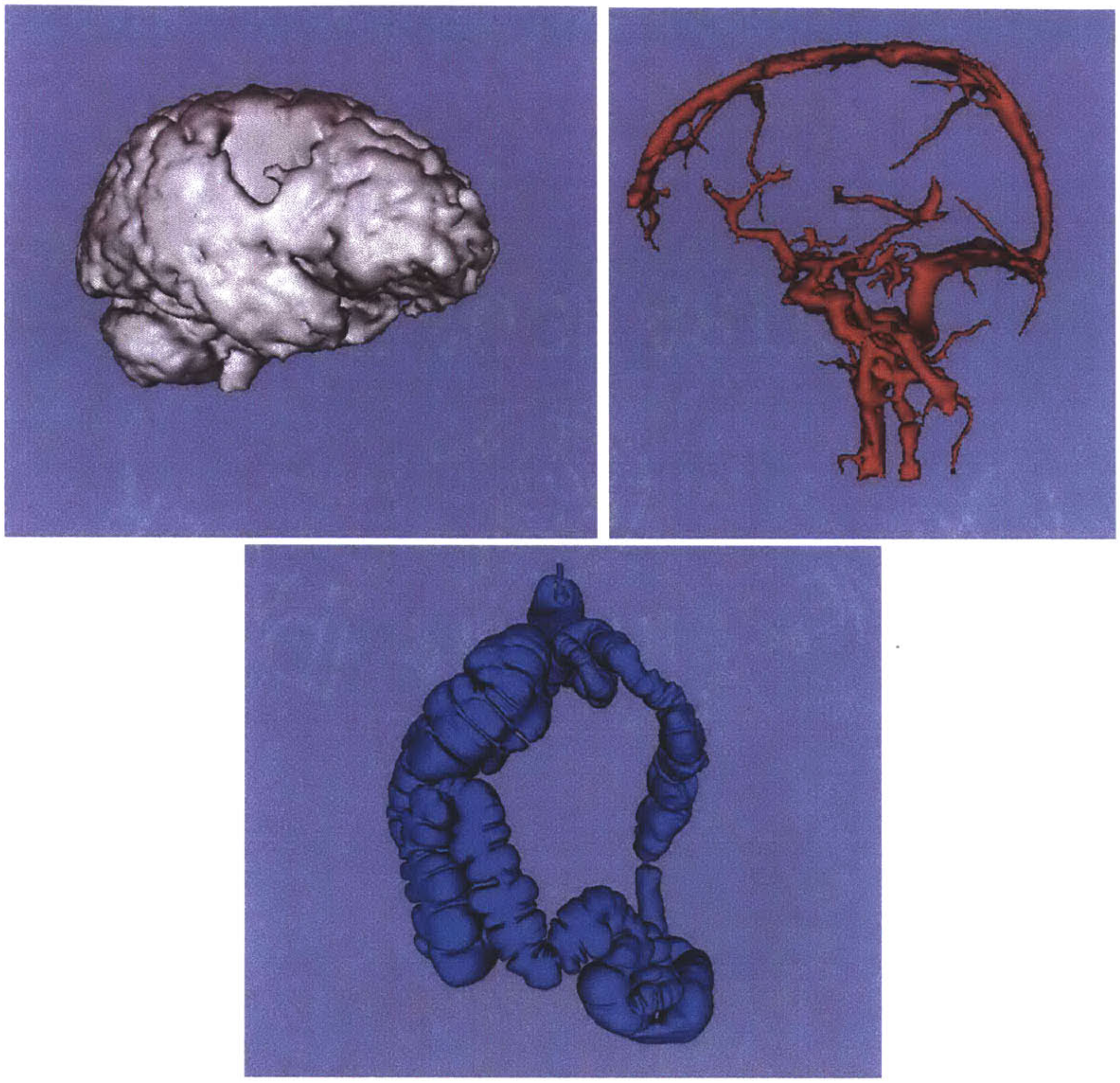


Figure 3-2: This Figure shows the three template models used to demonstrate our algorithm: a brain (left), brain vessels (middle) and a colon (right).

Dijkstra's algorithm. One contribution of our work is an efficient implementation of the algorithm using optimized data structures in order to work at interactive speed in our software.

Another important contribution is an efficient algorithm that outputs a labelmap from a closed 3D mesh in case the labelmap for a particular model cannot be used. To create a 3D model, users create a labelmap of that model by segmenting an original grayscale volume. The Marching cubes algorithm then creates a 3D mesh from that labelmap. In most cases the labelmap is saved, but sometimes the file is lost. Additionally, 3D meshes can be modified by users with cutting and scaling operations and these operations are currently not applied back to the labelmap, so the 3D mesh and labelmap data become misaligned. If a user decides to conduct a fly-through of a modified 3D mesh, our algorithm should be robust enough to find the centerline of the 3D mesh, without relying on the original labelmap. The solution is to re-create a labelmap from the modified 3D mesh, if needed, as a pre-processing step to the CPP algorithm.

We outline the steps of our algorithm in the rest of this Chapter. To test our algorithm on models of different topologies, we selected three template models shown in Figure 3-2: a brain for the spherical topology (left screen), a colon for a tubular topology (right screen) and vessels for a branching topology (middle screen). We will present the output of the algorithm on the three templates at each step.

3.2.1 Step 1: Create a Labelmap

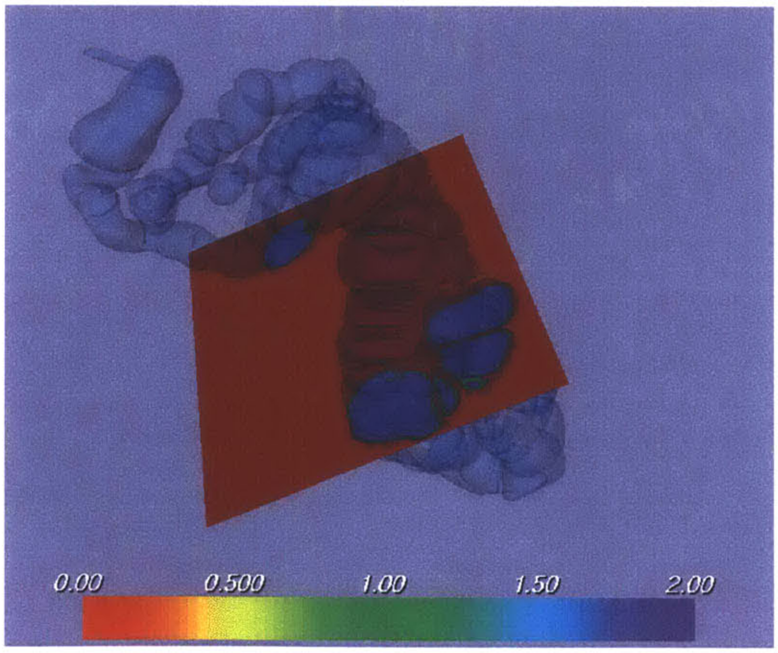


Figure 3-3: A labelmap of the colon shown on a reformatted slice

A **labelmap** is a volumetric datastructure where voxels that belong to an object have a particular label and the other voxels with the second label are called background voxels (meaning that they do not belong to the object). To create a labelmap from a 3D surface mesh, the first step is to find the **boundary voxels** of the object. The second step is to **'fill' the inside voxels** of the object. Figure 3-3 shows a slice through the labelmap of a colon where the boundary voxels have a value of 2 (green) and the voxels contained inside the colon have a value of 1 (blue). The rest of the voxels have a default value of 0 (red).

Our algorithm takes as input 3D meshes that are closed manifolds, that is if you were to put water inside the object it would not leak out. This is a good assumption for the 3D models that are created with the Marching cubes algorithm in 3D Slicer. There are rare cases, where some triangles in the mesh might be disconnected, but the crack is usually significantly smaller than any voxel, so it should not be noticed. In practice, we ran our algorithm on more than 40 models created in the 3D Slicer and never had to revise that assumption.

The most efficient way to label the boundary voxels is to scan each cell of the mesh and label all the voxels that the cell intersects. This keeps the cost of the algorithm to O(n) where n is the number of cells of the mesh. Since the user can choose the resolution by selecting different voxel sizes, this has to be taken into account when scanning the cell. In practice, we parametrize the triangle cell and step through the plane of the cell with a step proportional to the voxel size. At each step, we label the voxel that we are currently intersecting. The pseudo-code for this iteration is in Appendix A.

The result of this iteration applied to the brain dataset is shown at different heights of the model in Figures 3-4 and to the vessel dataset in Figures 3-5. The boundary voxels have a value of 2 (green).

The next step is to 'fill' the inside of the object. In order to do that, one would have to start from an inside voxel and grow a region from that voxel until the front of the growing region hits a boundary voxel. In practice, it is hard to find an inside voxel and guarantee that it is inside the model, so we inverse the problem and fill the outside (background) voxels instead. We pick the first voxel of the volume and guarantee that it is outside by padding our volume with an extra layer of voxels at

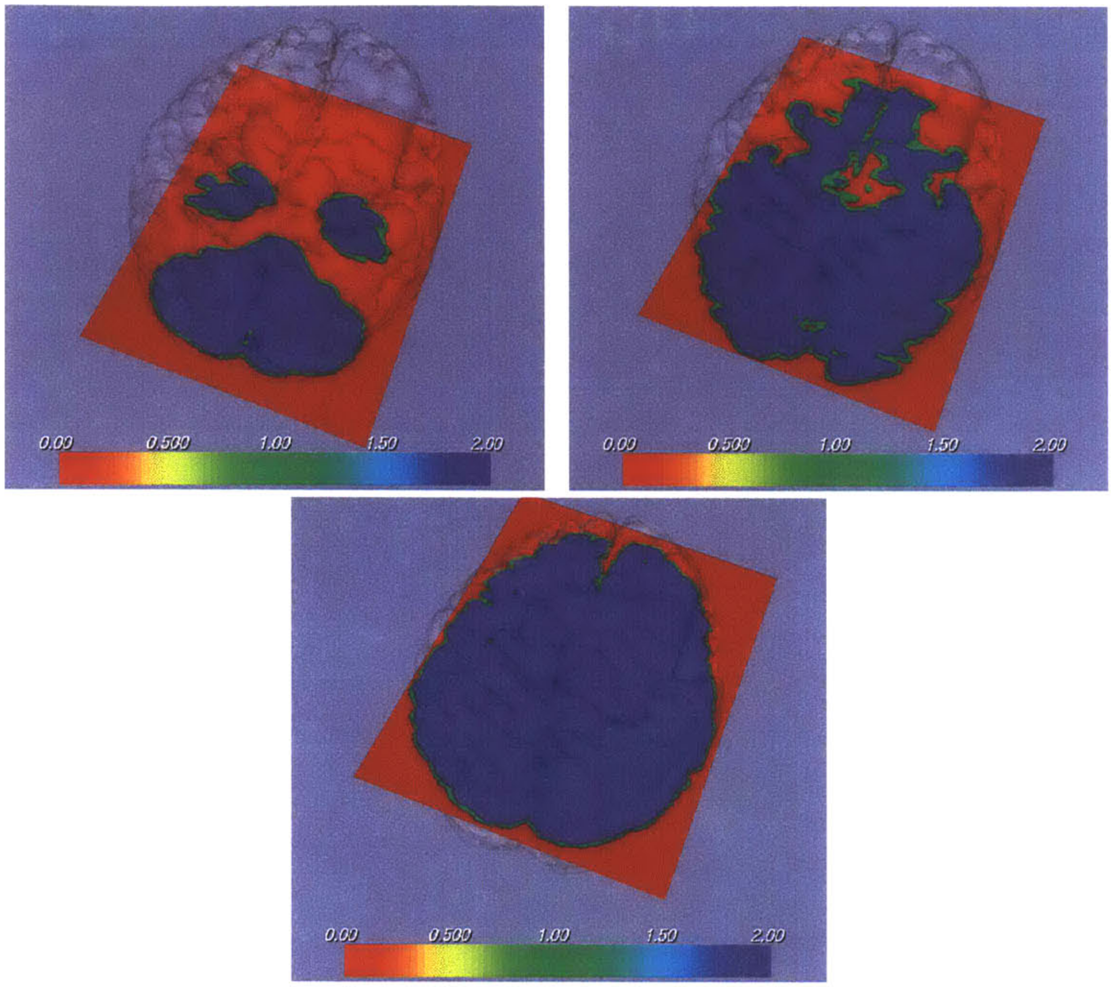


Figure 3-4: The result of the labelmap creation for the brain dataset at different heights through the model

initialization phase (see Appendix A). The pseudo-code for the filling algorithm can be seen in Appendix B.

The result of this iteration applied to the brain dataset is shown at different heights of the model in Figures 3-4 and to the vessel dataset in Figures 3-5. The inside voxels have a value of 1 (blue).

3.2.2 Step 2: Finding the Distance Map

The next step is to apply a **distance transform** to the labelmap. The aim of a 3D distance transformation is to compute the distance from a point p to an object O, i.e. to a set of voxels. A Euclidean Distance Transform (EDT) computes the exact

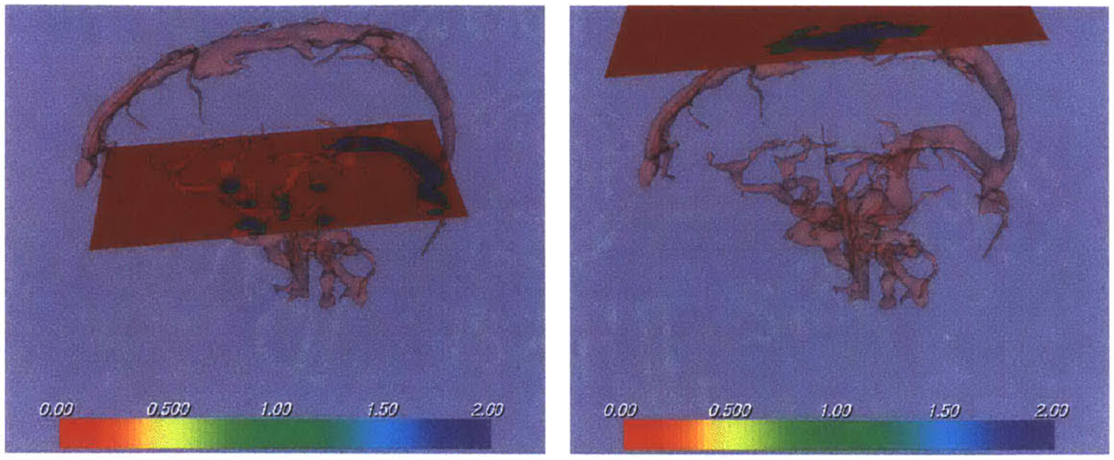


Figure 3-5: The result of the labelmap creation for the vessels dataset at different heights through the model

euclidean distance from p to O. In our case, O is the set of *boundary* voxels of the labelmap and the set of points p are the *inside* voxels of the labelmap. For this reason, we call it Euclidean Distance from the Boundary (EDFB) transform. Once we have computed all distance transform for every point p, we create a **distance map** that encodes that EDT value for every voxel of the original labelmap. *Outside* voxels are given a value of 0. Figure 3-6 shows a distance map for a 2D sketch of a colon where dark values are close to 0 and the lighter the pixel, the higher the EDT.

For our algorithm, we use the implementation of Olivier Cuisenaire's multi-resolution fast distance transform [32]. The algorithm first calculates a fast but approximate distance transform using a coarse neighborhood. A sequence of larger neighborhoods is then used to gradually improve the approximation.

The distance maps produced by the EDFB step of our algorithm are shown on the templates in Figure 3-7 and 3-8. The color bar at the bottom of each image shows the range of EDFB values and their color encoding for the distance map shown on the plane. As we can see on the figures, the range of values for the distance map of the vessels is far smaller than the one for the brain. This is expected since the vessel radius is much smaller than the radius of the brain.

Once we have the output of the EDFB, we effectively have a scalar field where the most centered voxels are the most positive voxels. For most topologies, the most

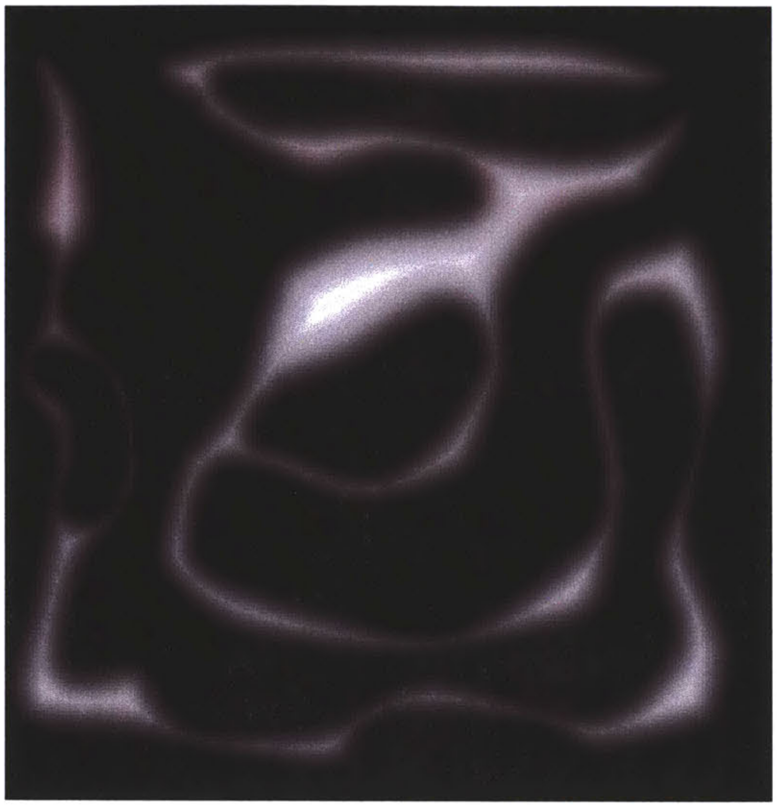


Figure 3-6: A distance map of a sketch of a 2D colon where black pixels are close to 0 and the lighter the pixel, the higher the distance

positive voxels lie on a 2D plane that does not necessarily span the whole model. For example, in the brain dataset of Figure 3-7, the most positive voxels in the middle image form an bean-shaped 2D plane.

3.2.3 Step 3: Graph Representation of the Distance Map

Since most models have a complex skeleton, extracting a centerline automatically with no user input is a difficult task. We propose a more user-friendly and efficient solution by prompting the user for a start and end point in order to avoid the additional complexity of extracting a skeleton and pruning its unwanted branches.

Once we know a start point and end point, we can extract centerline information from the distance map in a simple and efficient manner. Our solution is to create a **graph** that connects all the voxels contained inside the object and give edge weights to the node such that the path of least resistance from one voxel to another is to stay

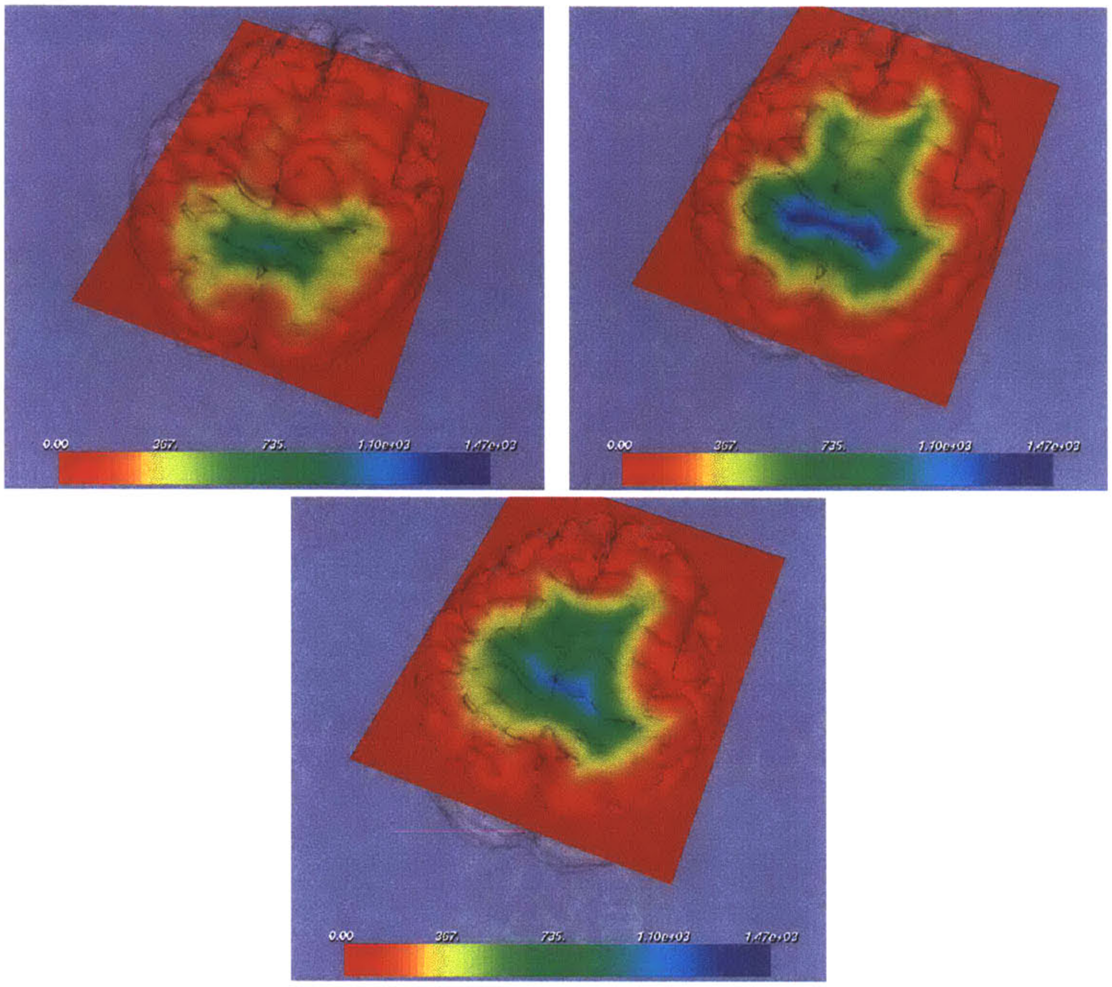


Figure 3-7: The result of the distance map creation for the brain dataset at different heights through the model

as centered as possible.

To create such a graph, all the non-zero voxels of the EDFB output become nodes. The edges between the voxels are bi-directional and the weight of an edge between node u and v is the inverse of the DFB of v. Intuitively, this means that we have a graph where voxels in the center have a low edge-weight and voxels close to the boundary have a high edge-weight.

The problem is then simplified to finding a shortest path between two nodes in the graph since the path of least resistance is along the most centered voxels that have a low edge weight. To find the shortest path, we use a variant of Dijkstra's algorithm. We first describe the standard Dijkstra algorithm and then present our variant.

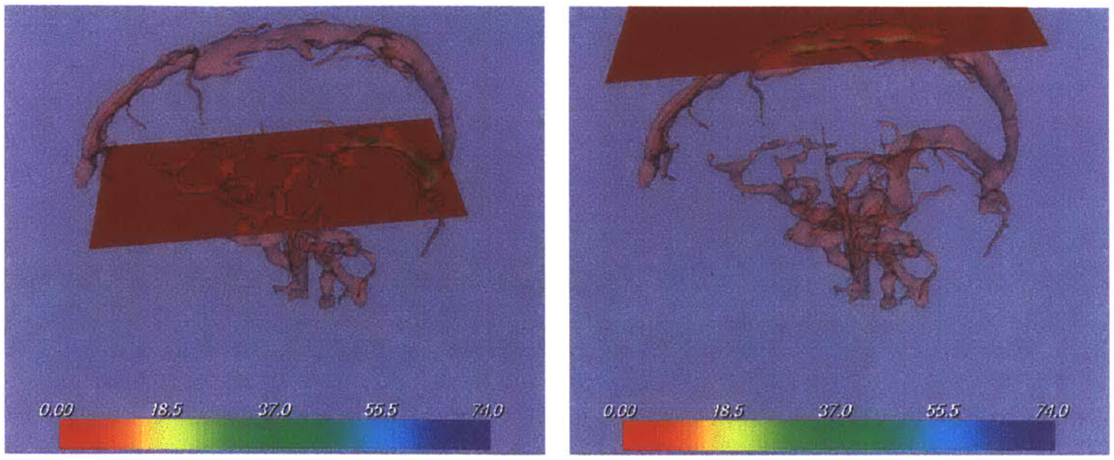


Figure 3-8: The result of the distance map creation for the vessels dataset at different heights through the model

3.2.4 Step 4: Extracting the Centerline

Dijkstra's algorithm is a single-source shortest path algorithm: it creates a minimum-cost spanning tree between a source point and every other node in the graph. A detailed algorithmic description of Dijkstra's algorithm can be found in Appendix C and in [33].

In 3D, Dijkstra's algorithm works by propagating a spherical front from the source point where each node visited by the front has been assigned an optimal path from the source to itself. The measure of optimality for a path, or the *cost*, is the cumulative weight of the edges traversed to go from the source to the node. We will first explain how the front is propagated and then how the optimal path is assigned.

During initialization, all nodes have a cost of ∞ except for the start node that is given a cost of 0 and all nodes are inserted in a priority list, called Q. In addition, each node points to another node, called its parent. At initialization, all nodes point to nobody.

At each iteration, the node u with the lowest cost is extracted from the Q and put into the *Visited* list. Then the node enters the expansion phase where the cost of its neighbors that are *not* in the Visited list is revised as follows:

- if the cost of the neighbor v is greater than the cost of the node *and* the edge weight to go from the neighbor v to the node u, then this means that we have found a more optimal path to go from the source to the neighbor v, via the node u. We therefore update the parent pointer of the neighbor v to point to the node u, as well as the cost of the neighbor v, which brings it up in the priority list.

- otherwise, do not update anything

The algorithm terminates once the Q list is empty and we have therefore visited all the nodes in the graph. Intuitively, we see that the front propagates continuously and symmetrically from the start point since it reaches nodes in the order of their distance from the start (their cost). Once the front has reached a node, the node knows its cost is optimal since all the future nodes hit by the front will have a higher cost. For this reason, Dijkstra is a greedy algorithm.

Once the algorithm terminates, it is straightforward to find a optimal path from any node in the graph to the source, by following the parent pointers from the node back to the source.

3.3 Optimizations

In this section, we describe some optimizations we made to our algorithm in order to run in interactive time without allocating too much memory. The optimized algorithm is described in appendix D.

3.3.1 Optimized Dijkstra for Virtual Endoscopy

Since our user gives us a specific end node, Dijkstra's algorithm calculates more than we need. In fact, when the end node is reached by the front, we can stop the algorithm since we have found the optimal path from the end node to the start point. The algorithm will then return a centerline path sooner, therefore increasing the interactivity of our system. We have ran some experiments on a colon dataset with 3 landmarks: L1 is at one end of the colon, L3 is in the middle of the colon centerline and L2 is at the middle of the L1-L3 portion of the centerline and L4 is

Fraction of full path length	.25	.5	1
Centerline extraction (in sec.)	6	15	37

Table 3.1: Performance results for three different path lengths in the same colon data

at the other end of the colon. We then ran our algorithm querying a path between L1 and L3 (half of the centerline), L1 and L2 (one quarter of the centerline) and L1 and L4 (the whole centerline). The result of the experiments are shown in Table 3.1. As we can see, the smallest the length of the path extracted is, the smallest the extraction time. This makes sense on the colon dataset since the topology is tubular. As long as the source is chosen close to an endpoint of the tube, the front propagated by Dijkstra's algorithm will stop as soon as it reaches the end point.

3.3.2 Optimized Memory Allocation

Some of the medical images that we used to test our algorithm are very big. For example a colon dataset has a total of 16 million voxels!! We noticed however that the actual voxels belonging to the colon (the inside voxels) are only 1.6 million.

In order to not overwhelm the memory of the system that will run our algorithm for huge datasets, we had to optimize the amount of space used by the datastructures:

1. We used a heap datastructure for the priority queue since it only uses O(n) memory space, where n is the number of nodes in the Graph.

2. We also reduced the number of arrays, by eliminating the array *d* that stores the cost since the information is already available from the priority list Q that stores the nodes indexed by their cost.

3. we did not store a Graph datastructure in memory since this would allocate an additional n bytes of space. Instead, we find the list of neighbors for each node on the fly, and only insert the neighbor in the priority list Q when one of its neighbors is being visited. Once a node is inserted in the Q, we check that it is not inserted again by keeping a flag. The only node that is inserted during the

initialization phase is the start node. From there, the rest of the graph nodes get put on the queue successively at each expansion phase. A revised version of Dijkstra's algorithm can be found in Appendix D.

These optimizations were beneficial since the algorithm can now process datasets of more than 15 million voxels on a simple home PC.

3.4 Important Parameters

In this section, we describe some boundary cases that we encountered and the fixes to our algorithm to handle these cases. We also discuss the importance of choosing a good edge weight when building our graph.

3.4.1 The "Crossover" case

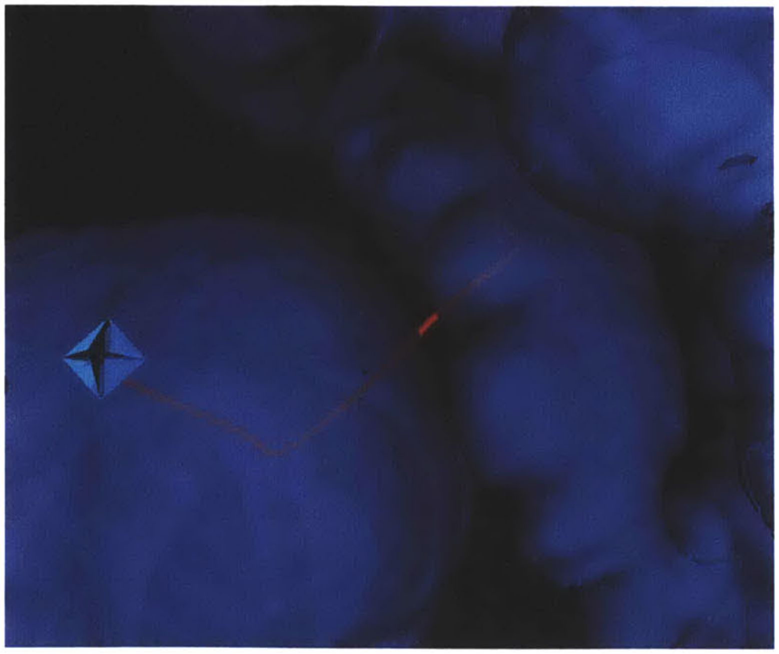


Figure 3-9: The "crossover" case when the path traverses two very close colon walls in order to "shorten" its path!

When we ran our algorithm on a particular colon dataset that did not have a labelmap, we noticed a strange behavior: at one end of the colon, the extracted path would "crossover" the colon wall and stop in the other end of the colon that is very close in space. This behavior is shown in Figure 3-9. After some investigation we realized the source of the problem.

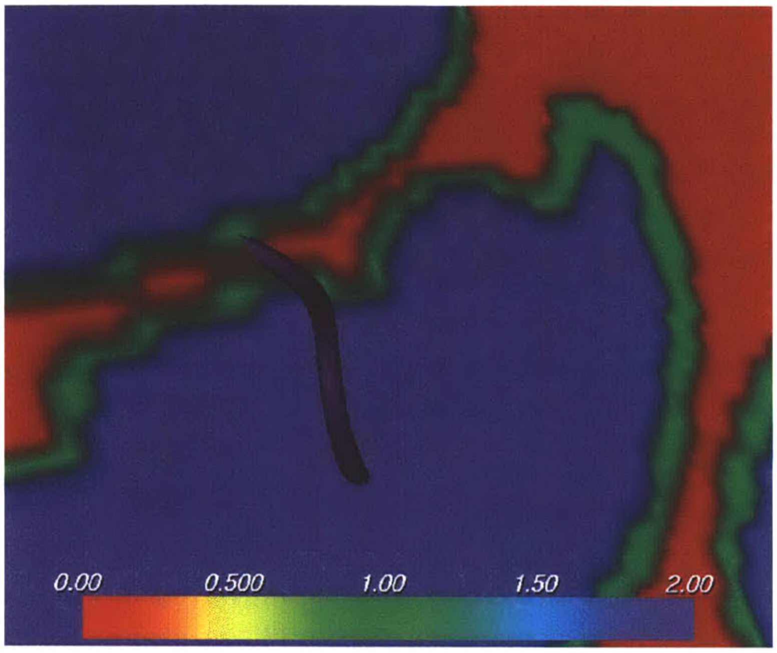


Figure 3-10: The "crossover" case shown on the labelmap when the path traverses two very close colon walls connected by the boundary voxels (in green, with a value of 2)

Since the two ends of the colon were so close in space (< 1mm) and the voxel resolution of our algorithm was 1mm, the boundary voxels of our reconstructed labelmap were actually "touching" each other. This meant that the two ends of the colon were connected in our graph representation, even if the connection was expensive since it involved going through two boundary voxels (voxels with the highest edge weight of 1). But since Dijkstra's algorithm tries to find a path with minimal cumulative weight, it was less expensive to cross over directly to the end then to go through the whole colon. One solution to this problem is to increase the resolution of our algorithm by choosing a smaller voxel size. This is not a feasible solution since decreasing the voxel size by n mm increases the size of the labelmap (the number of voxels) by n^3 !!! This means that we would lose a lot of interactivity. A better solution to keep an interactive algorithm is to highly penalize crossing over boundaries that are very

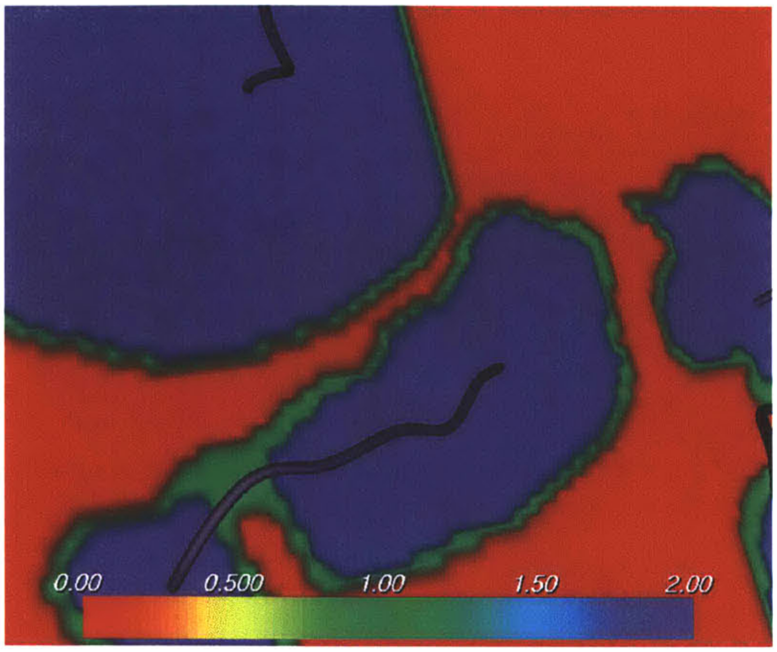


Figure 3-11: After the CPP algorithm was updated to handle the crossover case, the path does not cross over anymore

close to each other, by giving the boundary voxels a very high edge weight. The value of the boundary edge weight should be chosen such that no paths inside organs have a larger length. In practive we choose a boundary edge weight of 10,000mm since no organs in the body are longer than 10m (the colon is approximately 1 m to 1.5 m).

Figure 3-10 shows the path crossing over when the edge weight of the boundary voxels is not adjusted. The path crosses over on boundary (green) voxels. Figure 3-11 shows the same dataset after an adjustment of the boundary voxels' edge weight. The path does not crossover and is now going through the whole colon.

3.4.2 The Edge Weight parameter

In Section 3.2, we described that the edges between the voxels are bi-directional and the weight of an edge between node u and v is the inverse of the DFB of v. The edge weight is an important parameter that impacts on the shape of the path and how centered it is, as we have seen with the "crossover" case described above. In addition, Dijkstra's algorithm finds an optimal path, but necessarily the *only* optimal path if there are other possible paths with the same cumulative edge weigth.

We evaluated our algorithm with three different weight functions: $1/d$, $1/d^2$ and $1/e^d$, where d is the average between the EDFB value of voxel u and v. We chose those three functions to vary the gradient field of the graph. In other words, the function with the steepest slope, $1/e^d$, will have the strongest and steepest gradient variation from the boundary to the center of the object. With this function, we expect paths to stay in the center of the object because leaving the centerline has a very high cost. The function with the smallest slope, $1/d$, will have a smaller gradient variation from the boundary to the center of the object. This means that the path might sometimes leave the center of the object if neighboring nodes offer a better cumulative weight.

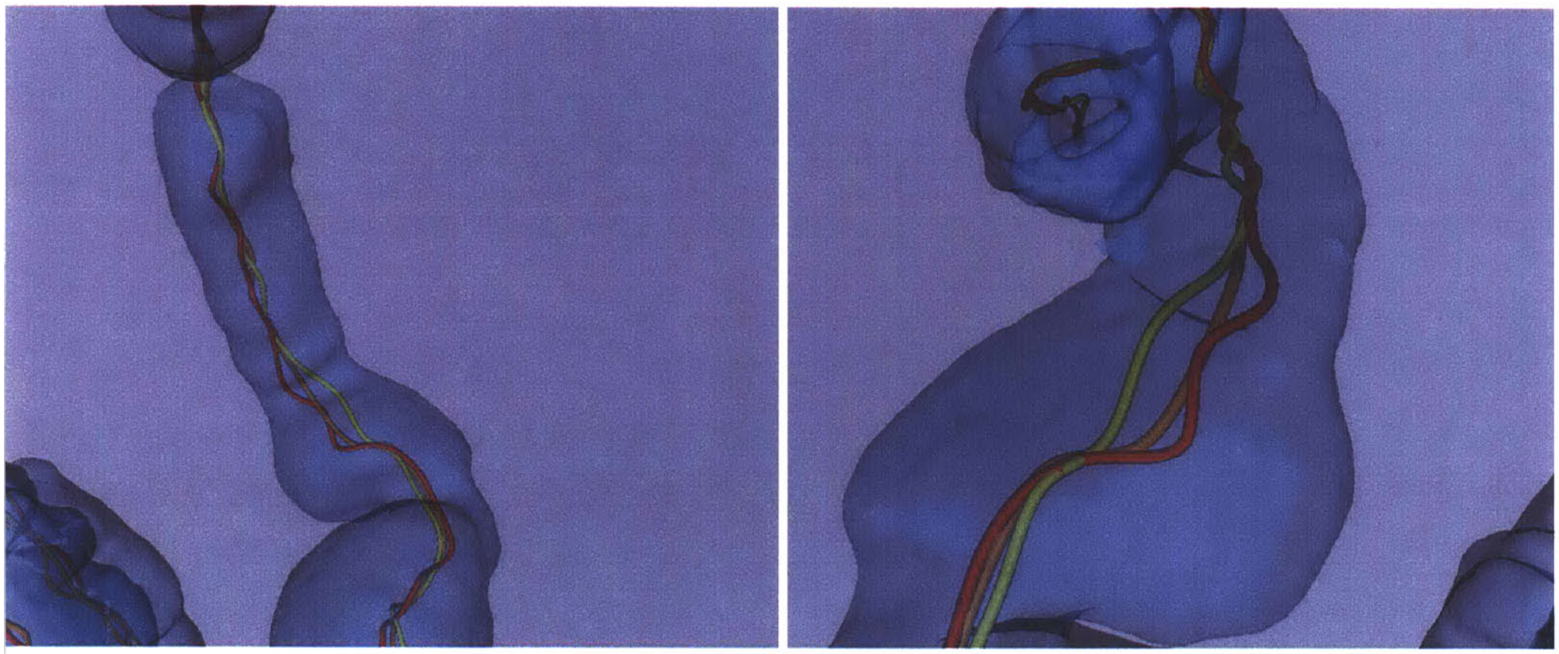


Figure 3-12: The output of our algorithm on a colon dataset when using three different weight functions. The yellow path is for $1/d$, the orange path is for $1/d^2$ and the red path is for $1/e^d$.

Figure 3-12 shows the output of our algorithm when using three different weight functions. The yellow path is for $1/d$, the orange path is for $1/d^2$ and the red path is for $1/e^d$. As expected, the edge weight function with a steeper slope, $1/e^d$, outputed the most centered paths. lIt is interesting to note, however, that the edge weight function with the least (linear) slope produces a smoother path. The tradeoff is as follows:

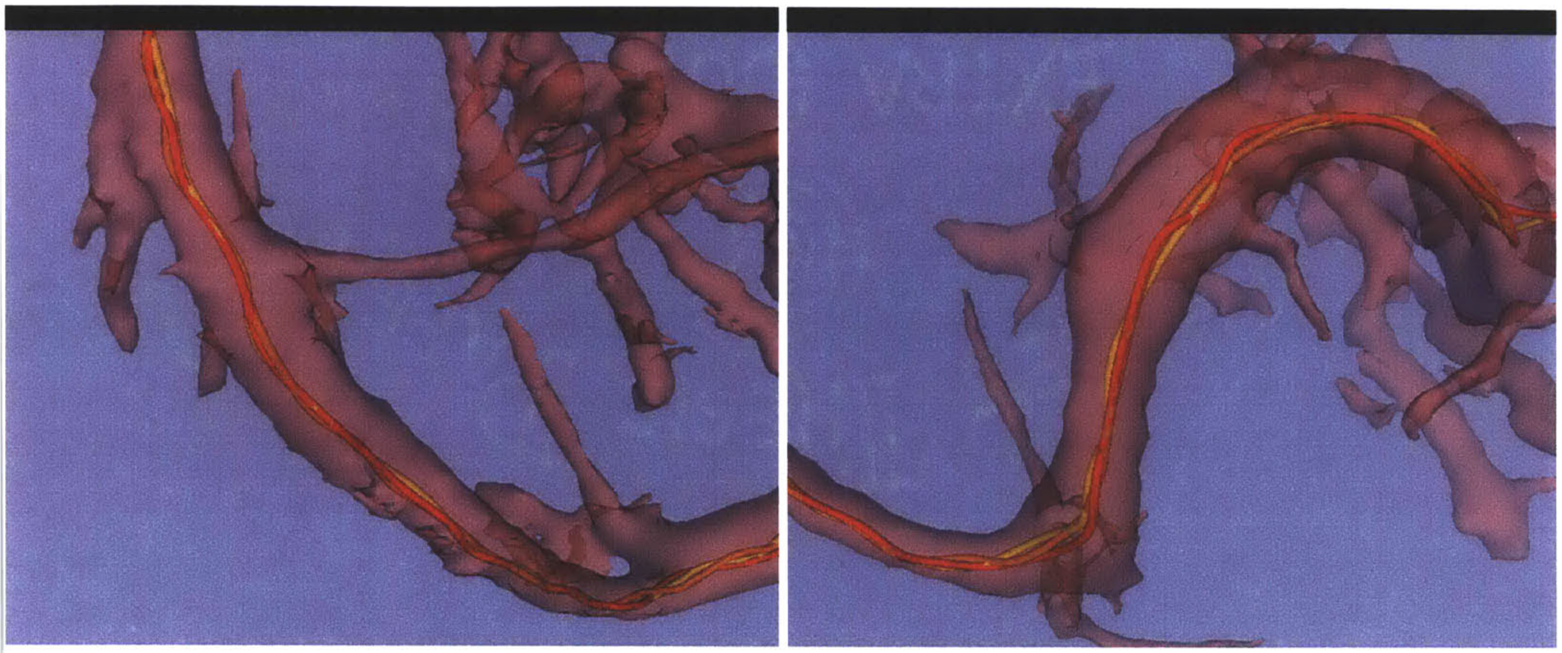


Figure 3-13: The output of our algorithm on a vessel dataset when using three different weight functions. The yellow path is for $1/d$, the orange path is for $1/d^2$ and the red path is for $1/e^d$.

- the most centered path will be produced with a edge weight function that has the highest slope (and therefore the strongest gradient) and therefore the path outputed will be very sensitive the topology of the organ wall.
- the smoothest path will be produced with an edge weight function that has the lowest slope (the weakest gradient) and will be less sensitive to the topology of the organ wall, and therefore might appear smoother.

Figure 3-13 shows a similar experiment for vessels. Since the radius of the vessels is much smaller, the edge weight functions produced much more similar results.

In the VE software, we chose the exponential gradient by default, but give the user the choice to change the edge weight function interactively.

3.5 Evaluation

In this section, we present some qualitative and quantitative evaluations of our algorithm.

Data sets:	brain	vessels	colon
Dimensions (total number of voxels)	2,736K	2,615K	16,105K
Graph Size (total number of inside voxels)	1,140K	89K	1,661K
Labelmap (in sec.)	1	1	17
Centerline extraction (in sec.)	24	2	37
Total Time (sec)	25	3	54

Table 3.2: Performance results of the CPP Algorithm for datasets of varying size

3.5.1 Quantitative Evaluation

In table 3.2 are shown the running times and statistics for the three different datasets on a Dell PC PentiumII 1GHz with 512MB of memory. As we can see, our algorithm runs at a very interactive rate for the vessels and the brain and at a slightly slower rate for the colon dataset due to its size. This is still an acceptable running time since most users won't mind waiting a minute if given update messages about the progress of the algorithm.

3.5.2 Qualitative Evaluation

The qualitative evaluations can be seen in Figures 3-16, 3-14 and 3-17. In Figure 3-14, we can see that the centerline is nicely centered even if the vessel tubes are narrow and branching. In Figure 3-16, it is interesting to see the behavior of the CPP algorithm on a spherical object, since a centerline cannot be extracted easily from its skeleton. As we can see in the left Figure, the path follows closely the topology of the surface by dipping when the surface dips as well. This is an expected behavior that proves that our algorithm behaves as expected. In Figure 3-17, the same colon is shown from the two opposite sides in order to fully see the type of centerline extracted from such a complex topology. Our algorithm behaves well, and the centerline is centered.

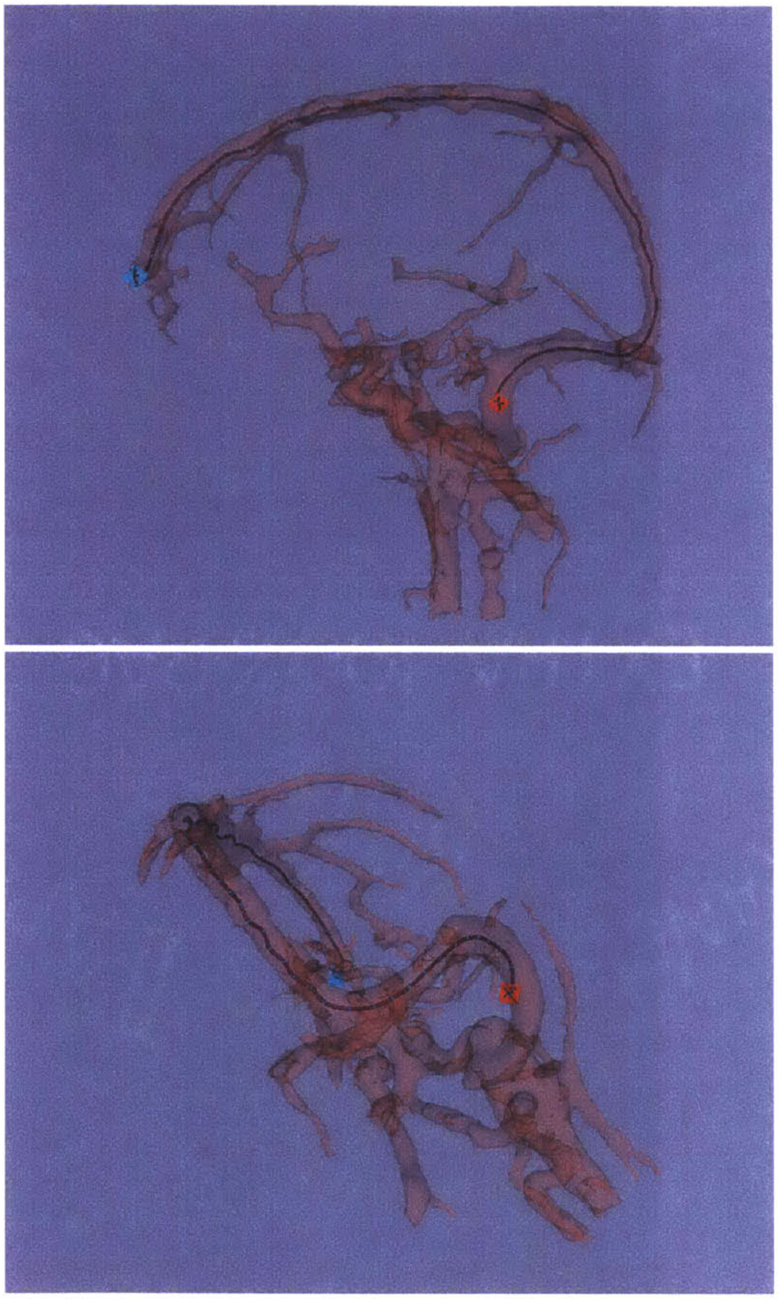


Figure 3-14: This Figure shows a centerline extracted between the two user defined points (diamond shaped) at two different angles.

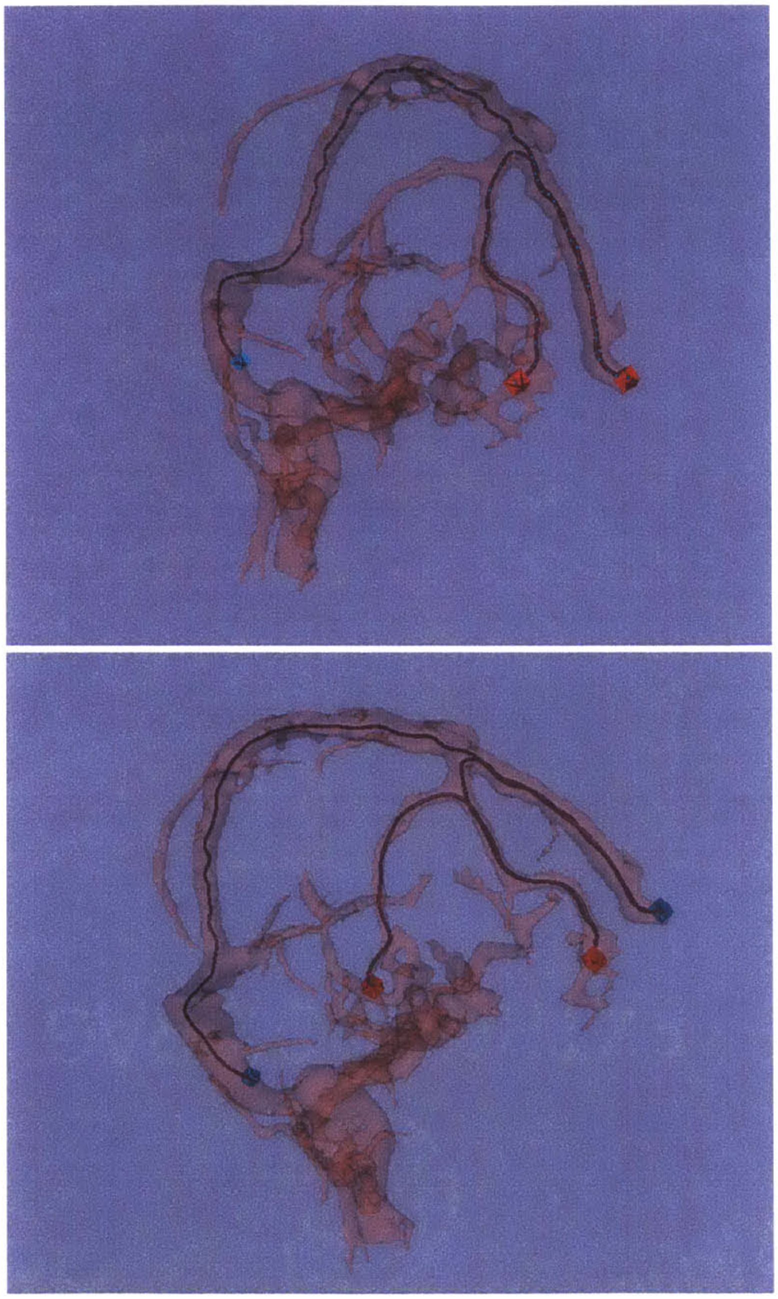


Figure 3-15: This Figure shows multiple centerline extracted between the user defined points (diamond shaped).

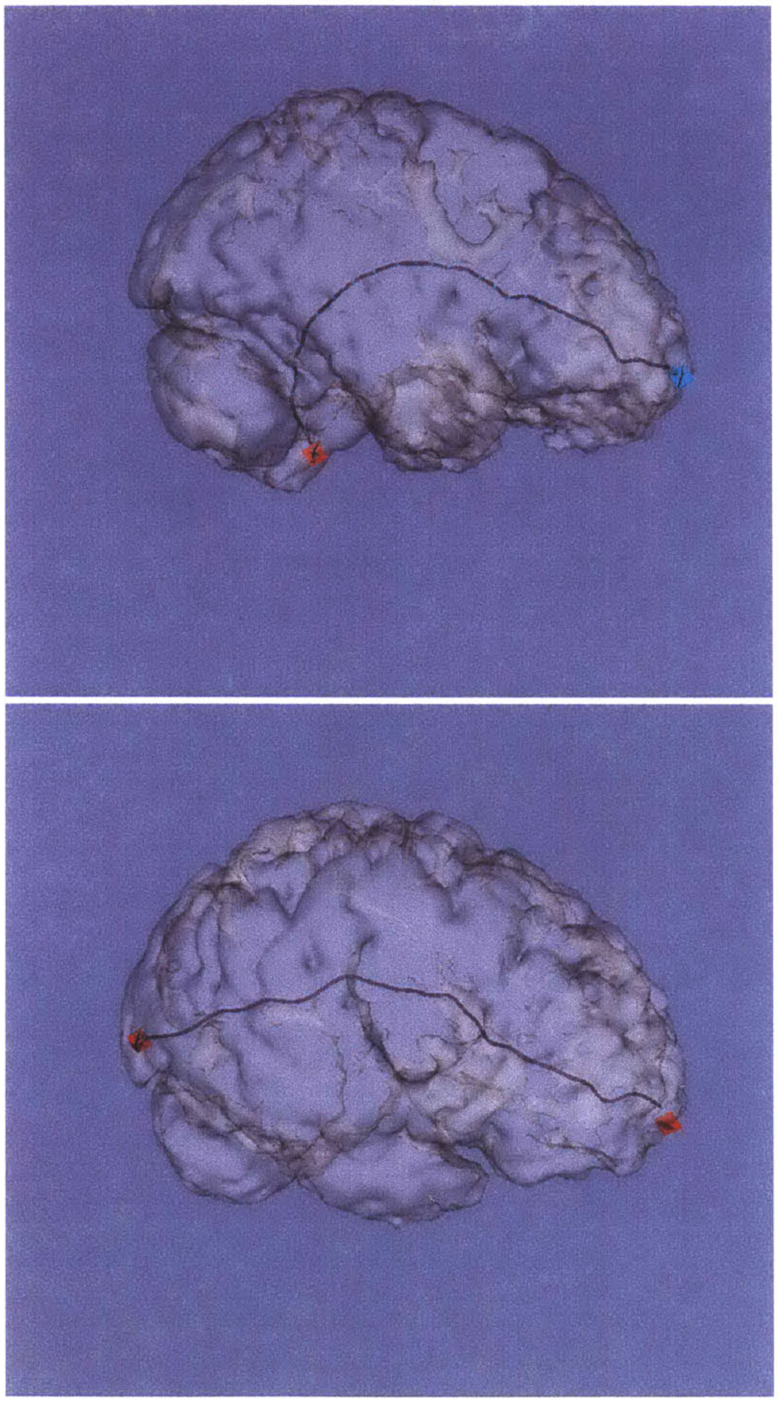


Figure 3-16: This Figure shows two different centerlines extracted between a different pair of user defined points (diamond shaped).

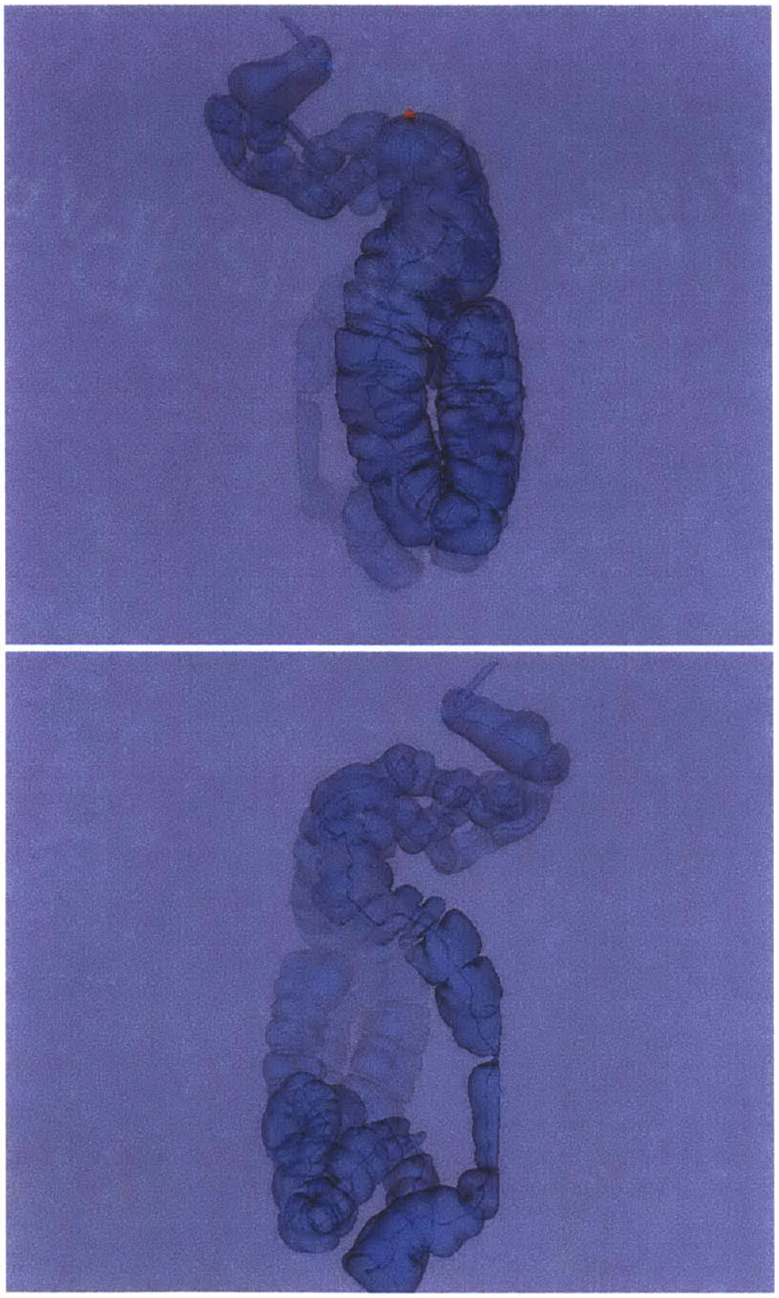


Figure 3-17: This figure shows the centerline extracted in the colon dataset in two different angles .

Chapter 4

Centerline Registration for Synchronized Fly-Throughs

In this chapter, we present an automated method for colon registration. The method uses dynamic programming to align data defined on colon center-line paths, as extracted from the prone and supine scans. This data may include information such as path length and curvature as well as descriptors of the shape and size of the colon near the path. We show how our colon registration technique can be used to produce synchronized fly-through or slice views.

4.1 Clinical Background and Motivation

In this Section, we review the state of the art screening methods for colon cancer and present a clinical motivation for our method.

4.1.1 Clinical Background

Colorectal cancer is one of the most common forms of cancer, and is associated with high mortality rates. Various screening methods used to detect colorectal cancers and pre-cancerous polyps are available, each with its own costs and benefits. In particular, fiber-optic colonoscopy is a well established and highly effective screening method, but is also invasive, expensive, time consuming and uncomfortable for the patient.

Alternative screening method are computed tomographic (CT) colon-ography and virtual colonoscopy. Recent work has indicated that virtual colonoscopy has the ability to provide doctors with the information needed to detect small polyps [34]. As CT technology has improved, providing higher resolution images obtained in shorter periods of time and with lower radiation doses to the patient, performing virtual colonoscopies have become more attractive as routine diagnostic procedures.

As described in Chapter 1, the presence of fake polyps, due to material and fluid left in the colon at the time of the scan can make the task of finding true polyps difficult. One solution is to obtain two CT scans of the patient, one with the patient in the prone position and one in the supine, and then compare both scans to differentiate artifacts (that should only appear on one scan) from polyps (that should appear on both scans) [6].

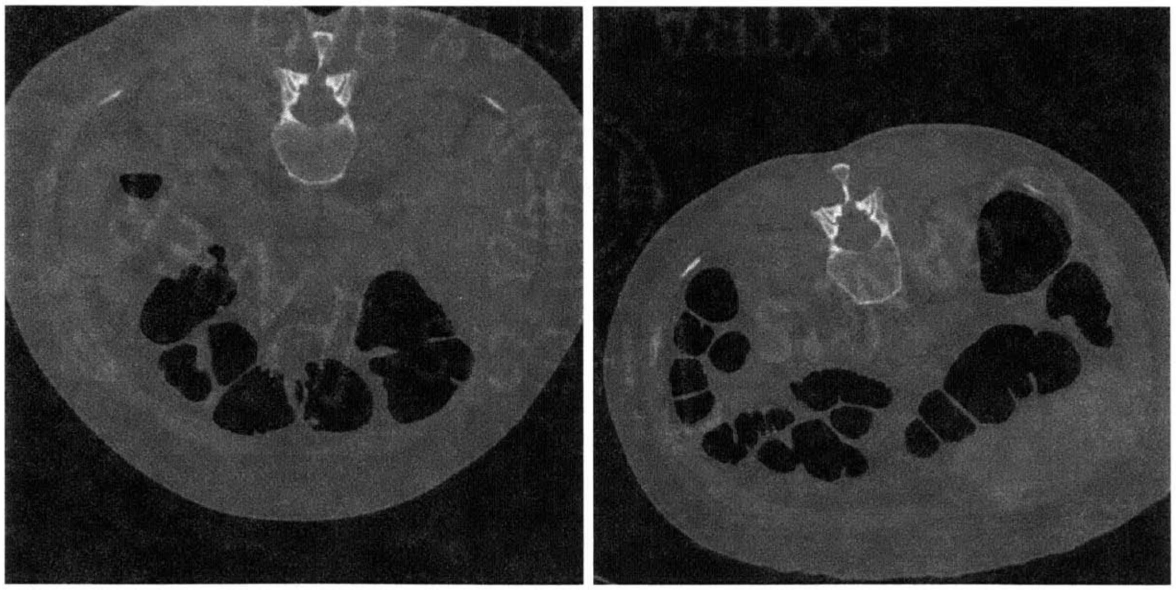


Figure 4-1: Axial slices through supine (left) and prone (right) scans. Although the same vertebra is pictured, the colon and other anatomical structures are not aligned.

Figure 4-1 shows an axial slice that intersects the same vertebra through both the prone and supine scan. As seen on the figure, the colon and other anatomical structures are deformed and shifted due to gravity and pressure factors, and look very different from one scan to the other. The deformation and shift between the colon and other anatomical structures is non-linear, so that surrounding structures are unlikely to provide enough information to match the two colons. An automatic volumetric deformable registration of one entire grayscale scan to the other would be desirable, but is an extremely difficult and time consuming task.

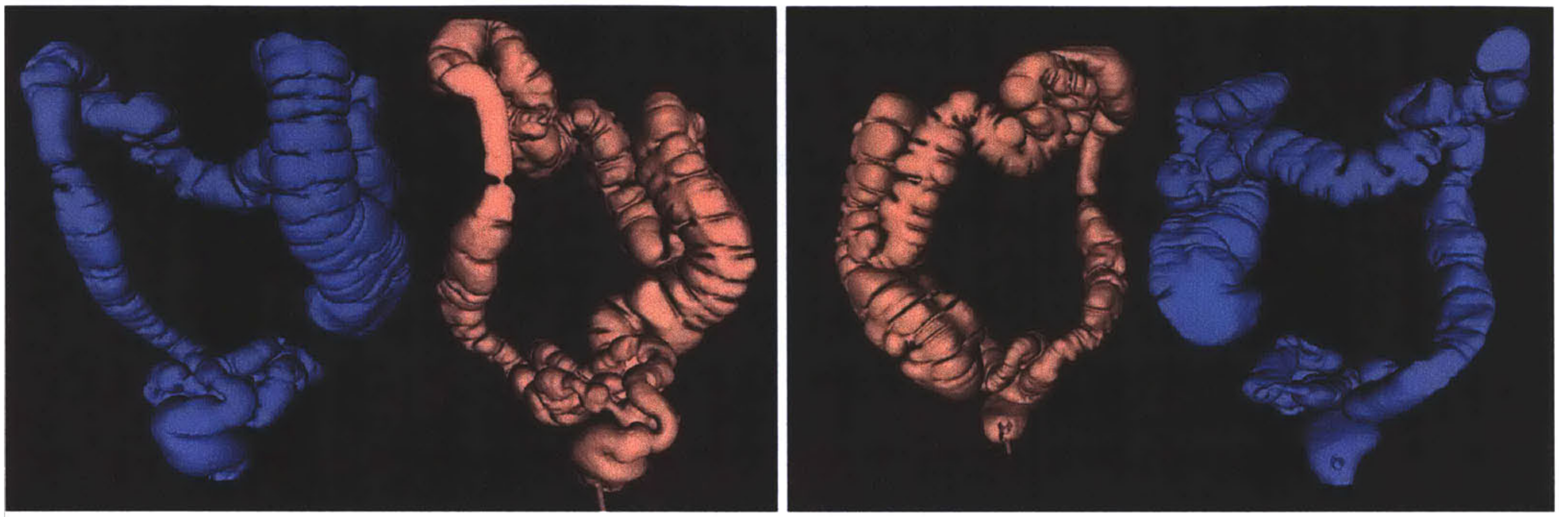


Figure 4-2: the supine (orange) and prone (blue) colons side by side respectively from the front (looking at the person's nose) and the back (looking at the person's back).

If one looks at the information in 3D, the non-linear deformation of both colons becomes very clear. Figure 4-2 show the supine (orange) and prone (blue) colons side by side respectively from the front (looking at the person's nose) and the back (looking at the person's back).

One solution to compare the datasets would be to show a synchronized virtual colonoscopy of both colons in order to directly compare the appearance of the inner surface at the same location in the colon. However, the centerline of both colons have a different shape, as well as a different length. Figure 4-3 on the right shows the centerlines for both the supine and prone colon (shown on the right). If a synchronized fly-through is to be performed, how do we simultaneously place the two endoscopes at each step and guarantee that each endoscope is placed at the same location in the colon? We propose the match the centerlines based on global geometrical information.

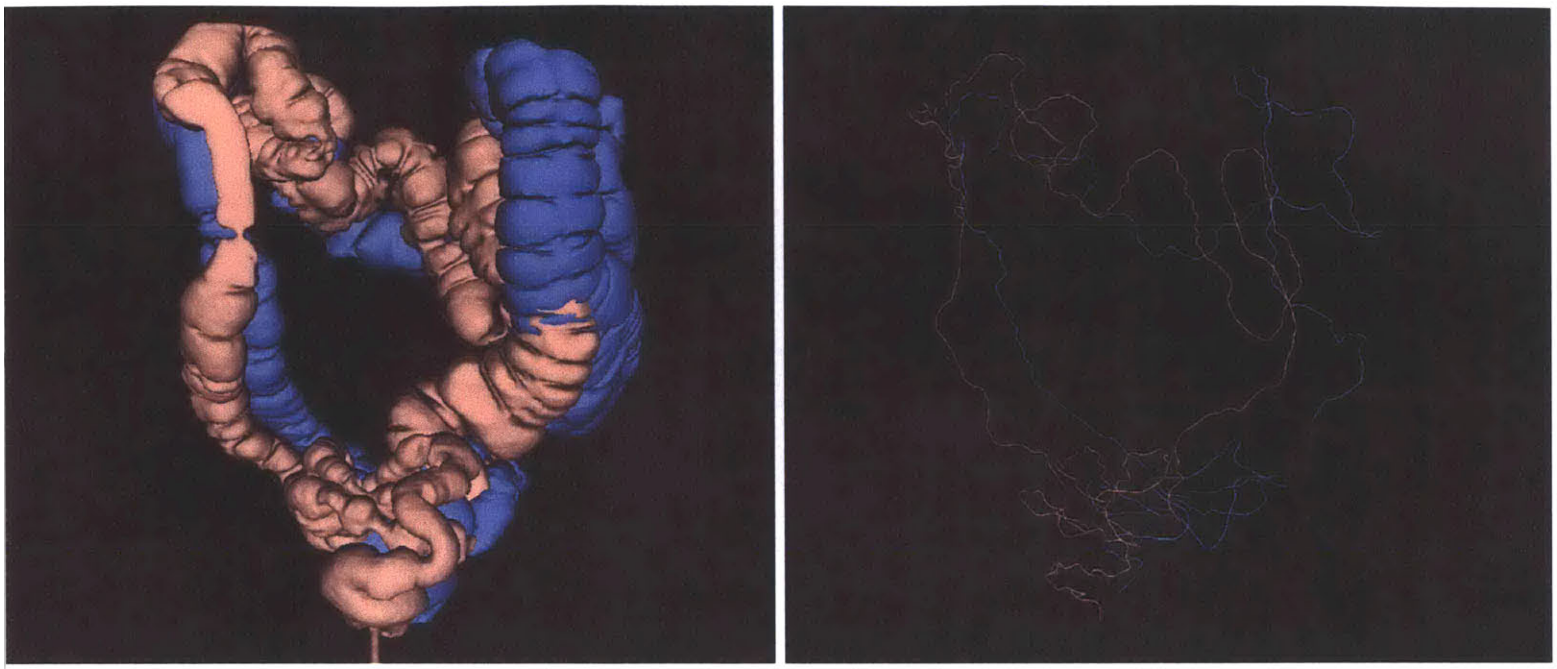


Figure 4-3: the supine (orange) and prone (blue) colons shown at the same location in space with their centerlines (right image).

4.1.2 Motivation for Our Approach

The centerline through both colons can provide meaningful geometrical information for colon registration, such as length and curvature. Centerline registration is also a simpler and less computationally intensive problem to solve than volumetric registration, since there is only one dimension to match. However, using only centerline information such as length and curvature might not be enough for a robust solution in cases where the colon is severely stretched and deformed between the two scans.

In this case, other important geometric quantities could help the centerline matching, such as radius, circumference or surface curvature information. To resolve this issue, we have developed a dynamic programming algorithm for 1-dimensional point matching, and we incorporate 3-dimensional surface information into our metric, or objective function, to match each point of the centerline. Dynamic programming has the advantage of being a fast and efficient algorithm that finds a globally optimal matching of centerlines, with respect to the objective function, while preserving centerpoint ordering [33].

In Section 4.2 we review the related work for registration of prone and supine datasets. In Section 4.3, we describe our centerline registration method using dynamic programming. Section 4.4 presents our results, including the use of the matching technique for synchronized virtual colonoscopy.

4.2 Related Work

Some commercial tools used to view CT image slices have an option to display a supine and prone scan side-by-side, with the option to flip the prone scan, so that images are presented in the same orientation. Effectively, the clinician has to register the two volumes manually in order to find interesting corresponding landmarks.

Acar [35] *et al.* have developed an automatic method that registers the medial axis of supine and prone colon surface models using linear stretching and shrinking operations. They sample centerlines of both colon models at 1mm intervals and examine the inferior/superior coordinates of path points to find local extrema and path inflection points. In order to find corresponding points, they linearly interpolate between inflection points using path length information.

This method only takes into account the local extrema located on the inferior/superior axis. If the colon shifts obliquely when the patient changes position, then some local extrema may not be accounted for. Further, the method does not allow for colon surface information to be taken into account. Information such as radial distance from the medial axis or circumference cannot be used in the matching process. Our method by contrast, is designed to address these points.

4.3 Registration Methodology

In this section, we present our method for the registration of colon centerlines extracted from prone and supine colon scans. Our method uses dynamic programming and geometric information to find an optimal match between sampled centerpoints. Once the centerlines are matched, fly-throughs are generated for synchronized virtual colonoscopy by stepping incrementally through the matched centerpoints.

4.3.1 Centerline Extraction

For centerline extraction of both supine and prone models, we use the CPP algorithm described in the previous chapter. Figure 4-3 shows the resulting centerlines.

4.3.2 Dynamic Programming

Overview

Once we have the centerpoints extracted for both colons, we match them using dynamic programming. Dynamic programming solves optimization problems by finding and recursively combining the optimal solutions to subproblems. A dynamic programming algorithm is efficient since it solves every subproblem only once and caches the solution, thereby avoiding the work of recomputing the answer every time the subproblem is encountered [33].

Dynamic programming has been used in a variety of contexts including for DNA and protein sequence alignment [36] as well as special cases of pose estimation in object recognition [37].

Registration Algorithm

In our case, we wish to align two sets of centerpoints, P_1^N *i.e.* the centerline of the prone colon containing the points indexed from 1 to N and S_1^M, the centerline of the supine colon indexed from 1 to M. The subproblems of this optimization task are all the pairwise matching of the subsequences of $P_1^i, (1 \le i \le N)$ and $S_1^j, (1 \le j \le M)$. We now describe the two steps of our centerline registration algorithm.

1. Recursive Definition of an Optimal Solution

   Let $f(P_i, S_j)$ be a cost associated with matching centerpoint P_i with S_j.

   Let us further assume that we already have an optimal minimal cost solution for the matching of the pairs of subsequences (P_1^i, S_1^{j-1}), (P_1^{i-1}, S_1^{j-1}), and

(P_1^{i-1}, S_1^{j-1}). If we define F to be a metric that evaluates the matching of the argument subsequences, we can find the optimal alignment of the centerlines (P_1^i, S_1^j) by solving the recursion:

$$F(P_1^i, S_1^j) = f(P_i, S_j) \ + \ \min \begin{cases} F(P_1^i, S_1^{j-1}); \ \text{(expansion at } P_i) \\ F(P_1^{i-1}, S_1^{j-1}); \ \text{(no expansion/compression)} \\ F(P_1^{i-1}, S_1^j); \ \text{(expansion at } S_j) \end{cases} \tag{4.1}$$

With this recursive expression, we fill in an $N \times M$ matrix with entries at $(i,j) : F(P_1^i, S_1^j)$, along with pointers in the direction of the preceding sequence matching which led to this optimal value. It is important to note that dynamic programming allows many-to-many mappings, resulting in mappings that can be locally compressions or expansions. For example, if $F(P_1^i, S_1^{j-1})$ is chosen as the optimal subsequence of $F(P_1^i, S_1^j)$, then centerpoint P_i will match to both points S_{j-1} and S_j, which would mean that in the locality of point P_i, the matching to the sequence S_1^j is a compression. We chose to penalize the amount of stretching and compression allowed with a penalty function $g()$ added to the first and third line of equation 1. We experimented with different values of $g() = 0.0$ and $g() = 0.1$ and experimentally found that the latter gives us a better match.

2. Extracting the Sequence Alignment

   By following the pointers from entry (N, M) to entry $(1, 1)$, we obtain a sequence of (i, j) pairs that define a point-to-point correspondence between point P_i and point S_j.

Objective function $f(P_i, S_j)$

As mentioned in Section 4.2, we use both centerline and geometrical information to give a value to each centerpoint.

- The centerline information is the distance from the first centerpoint to the current centerpoint normalized to the total length of the centerline.
- The geometrical information is the average radial distance from the centerpoint to the surface loop centered at the centerpoint.

The objective function $f(P_i, S_j)$ evaluates how closely two centerpoints P_i and S_j match. We have defined it as:

$$f(P_i, S_j) = \alpha(r_i^P - r_j^S)^2 + (1 - \alpha)(d_i^P - d_j^S)^2 \tag{4.2}$$

where r_i^P is the average radial distance at the ith centerpoint in the prone scan, d_i^P is the distance along the path of the centerpoint, and similarly for r_j^S and d_j^S. The parameter α is used to balance the two terms of the functional. The results for the colon registration presented in Section 4.4 are for $\alpha = 0.5$. Other functionals incorporating other surface information can also be used.

4.3.3 Synchronized Virtual Colonoscopy (SVC)

In order to visualize synchronized colonoscopies, we use our VE software to displays the two surface models and the location of both virtual endoscopes relative to the surface models, as well as the views of each virtual endoscope updated simultaneously to show the same location in the colon (see Figure 4-5). In addition, the user has the option to display a reformatted CT slice through each volume that moves along with the virtual endoscope and stays parallel to the view plane. This functionality allows the user to compare the grayscale image registered to match the colon data.

4.4 Results

Here we show the results of our algorithm applied to supine and prone CT scans taken of the same patient 8 minutes apart. Both scans had 512 matrix size with slice thickness of 1mm, and 362 slices. Pixel size was 0.6mm.

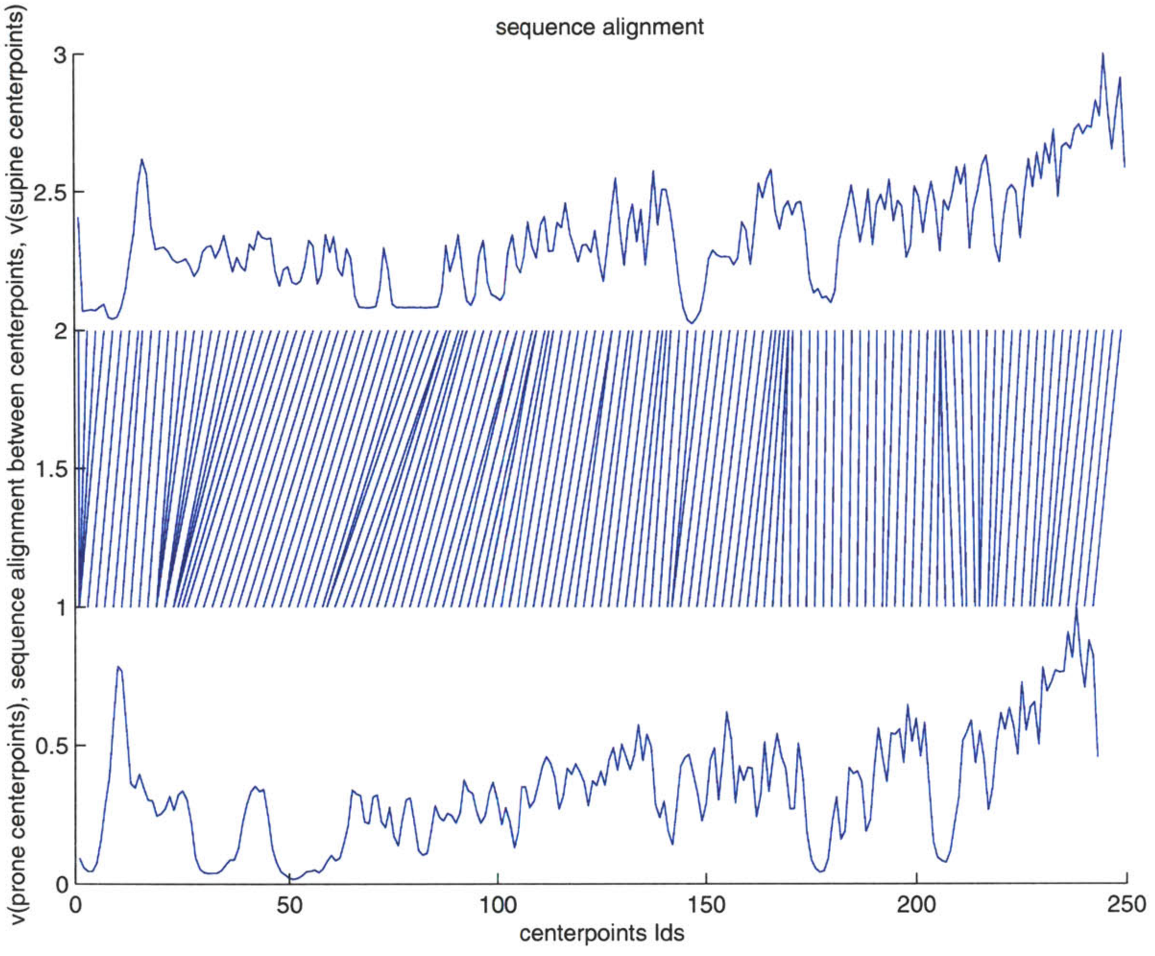


Figure 4-4: Sequence alignment of prone and supine centerpoints.

Figure 4-4 shows an objective value with $\alpha = 0.5$, plotted for both colons as well as the sequence alignments found by our algorithm between points using a penalty for excessive stretching or compression (we used $g() = 0.1$). Each line in the middle of the figure shows a correspondence between a prone and supine point. As can be seen, there are areas of slight expansion and stretching that are detected.

In order to have a preliminary evaluation of the fly-throughs produced by our algorithm, we recorded how many frames matched for different values of α out of the

α	% of matched frames
0	40
0.5	94
1	88

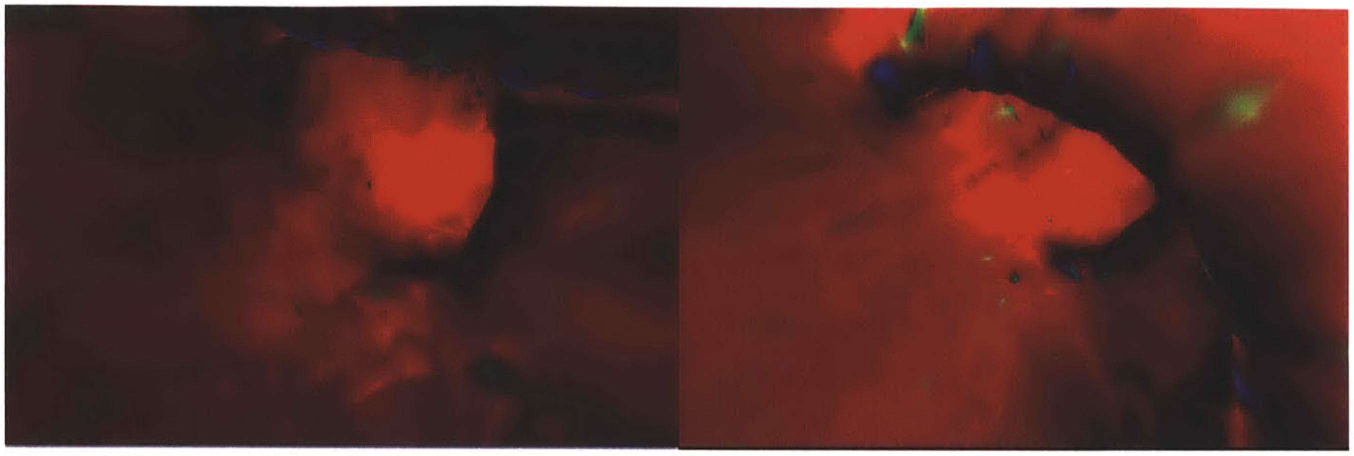
Table 4.1: Performance results with different objective functions

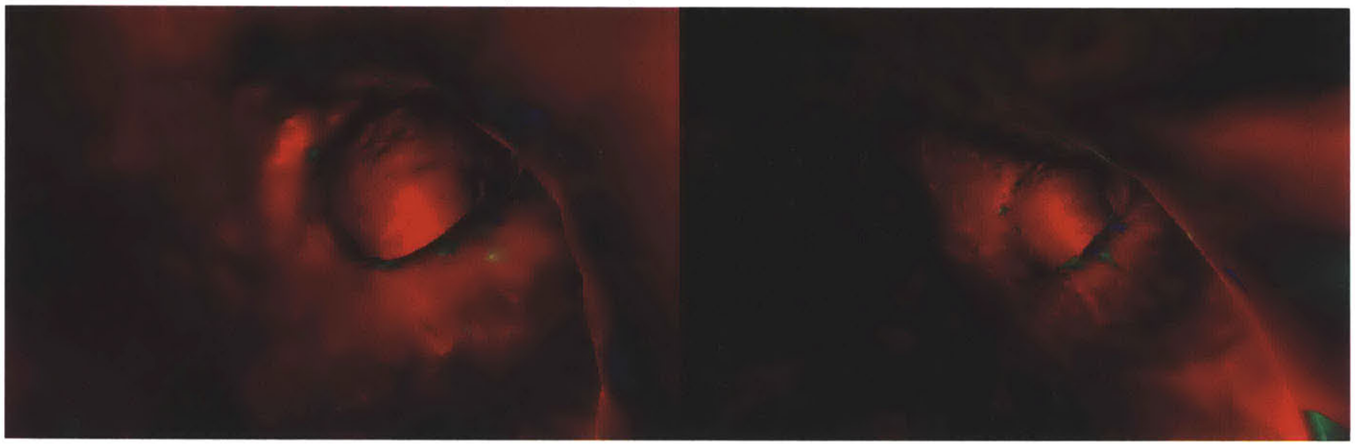
total number of frames (278). These results are presented in Table 4.1.

From the results, we see that using the distance metric alone ($\alpha = 0$) fails because the initial 6 centerpoints of the supine centerline do not exist on the prone centerline. This causes a misalignment of frames throughout the fly-through. Using average radius information alone ($\alpha = 1$) is considerably better, except at particular centerpoints where there is a collapse of the colon due to fluid leftovers. This causes a temporary misalignment until the next correct radius information is matched. We found that a combination of both metrics is optimal and gives us a 94 % frame match rate.

Figure 4-5 presents some of the frames of the synchronized virtual colonoscopy performed on the supine (left screen) and prone (right screen) models. The first four frames show a close match where one can recognize similar landmarks in both views. Frame 54 show a close match, however a few frames later at frame 66, the prone colon shows an obstruction while the supine colon does not. We can therefore assume that the obstruction is an artifact, probably some left-over fluid that stayed at this area of the colon was it was scanned in the supine position . Frame 75, shows again a new close match between the two views.

These results are encouraging and we are in the process of evaluating our algorithm with a colonoscopist at the Brigham and Womens hospital to conclude whether this method could be used clinically.


(frame 20)


(frame 26)

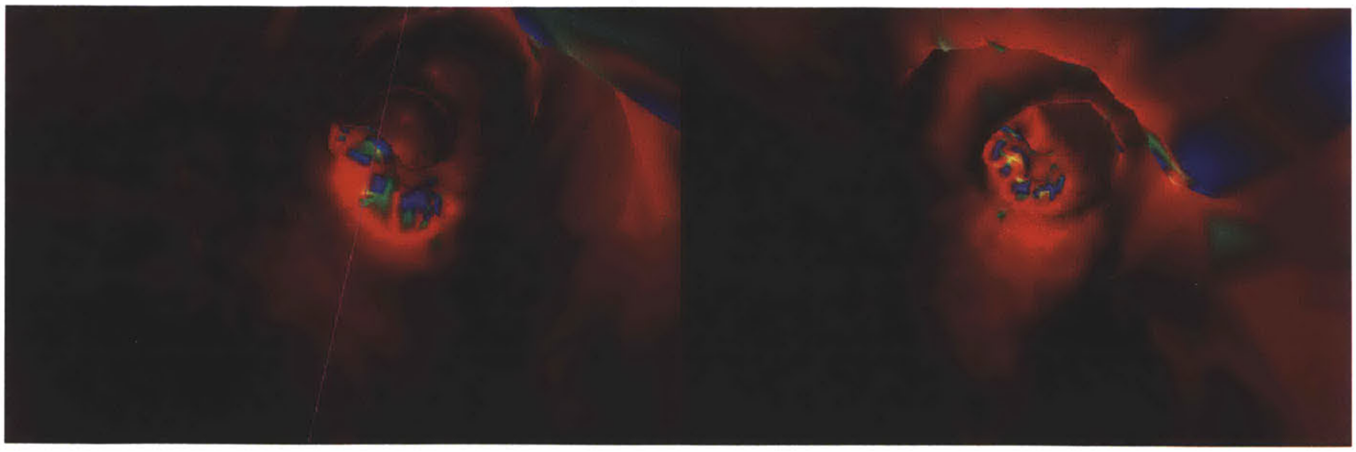
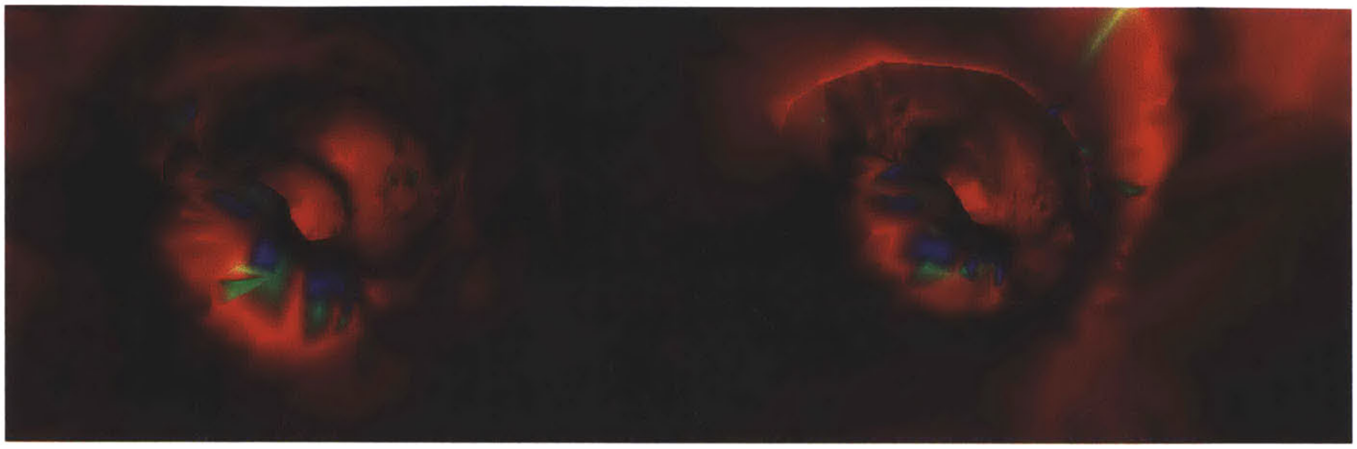
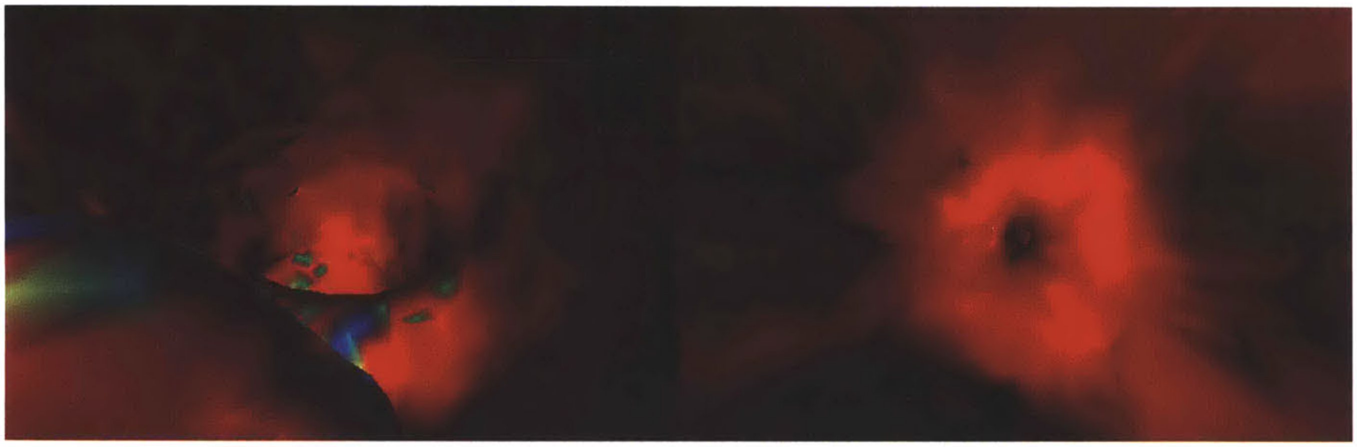

(frame50)

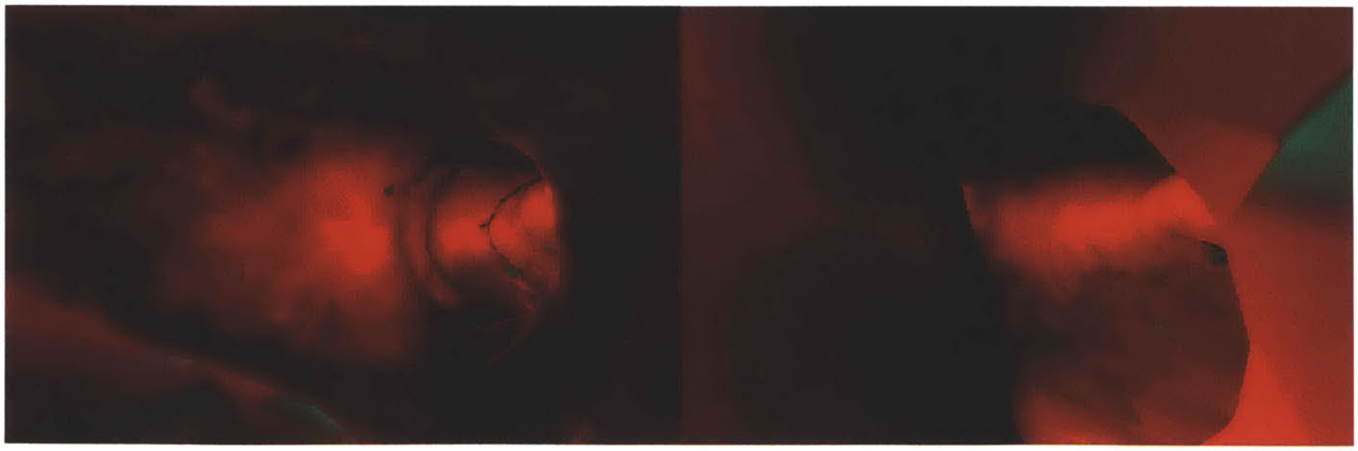
Figure 4-5: Different Frames of the Synchronized Fly-Throughs



(frame54)



(frame 66)



(frame 75)

Figure 4-6: Different Frames of the Synchronized Fly-Throughs

Chapter 5

Conclusion

This thesis describes a valuable system for virtual endoscopy. Our system can be used for manual, semi-automatic or automatic exploration of 3D medical data. The system is easy to use for beginners who wish to extract a trajectory path automatically and contains more sophisticated features for advanced users such a s volume reformatting and centerline registration of non-linearly deformed colons.

The VE system has been used as an analysis and educational tool for various cases at the Brigham and Womens hospital in Boston.

The system's open-source platform will allow future researchers to further advance the field of virtual endoscopy by integrating their algorithm into the VE software.

Appendix A

Production of Volume Data From a Triangular Mesh

The algorithm described in this appendix takes as input a 3D triangulated mesh and converts the mesh to a volume representation where voxels that represent the mesh are labeled “boundary voxels”. The output volume has the dimensions of the bounding box of the 3D mesh.

In this particular implementation, the algorithm outputs a volume where boundary voxels have a value of 1 and all other voxels have a value of 2.

```
initialize_labelmap(Mesh M)

   Spacing: size of a voxel edge set by the user
   dim[3]: bounds of M          // dim is a column vector that contains the
                                // dimensions of the bounding box of the mesh
    Origin[3]: origin of bottom
             back right corner
             of M

    V : [dim[0]+1 x dim[1]+1 xdim[2]+1 ]   // V is a 3D array of voxels
```

```
                                        // pad the volume by 1 voxel in
                                        // each dimension for the boundary
                                        // fill algorithm
    V.origin: Origin                    // V.origin are the coordinates
                                        // of the the first voxel of V

    foreach voxel v in V
          v.value = 2



create-labelmap(Mesh M)
     initialize_labelmap(Mesh M)
     foreach triangle in M
          extract the coordinates of its 3 vertices: v0,v1,v2
          //parametrize triangle
          stepS = Spacing / (2 * distance(v0,v1))
          stepT = spacing / (2 * distance(v1,v2))
          for(float t = 0; t <= 1; t = t+stepT){
             for(float s = 0; s <= 1; s = s+stepS){
                // check that we are still inside the triangle
                if ((s >= 0 && t >=0 && (s+t)<= 1))
                insidePoint = v0 + s (v1 - v0) + t (v2 - v0)
                coord = floor(insidePoint - Origin / Spacing)
                find voxel v of volume V at coordinates coord
                v.value = 1
```

Appendix B

Filling Algorithm Step

This 3D filling algorithm was adapted from a 2D filling algorithm taugth in the MIT Computer Graphics course. The algorithm takes as input a volume where boundary voxels are specially labeled. The assumption is that the boundary voxels are a reprensation of a closed object. The algorithm outputs a volume of the same dimensions that labels differently the voxels outside the object, the boundary voxels and the voxels inside the object.

The algorithm performs a propagation that starts at the bottom right back corner of the volume and expands a front until it hits a boundary voxel (labeled 1 for this implementation). While the front expands, it gives a new label of 0 to all the outside voxels it encounters (originally labeled 2 for this implementation).

So in summary, the particular implementation of this algorithm takes as input a labelmap where the encoding is the following:

```
outside label: 2 , boundary label: 1 , inside label: 2
```

and outputs the following labelmap:

```
outside label: 0 , boundary label: 1 , inside label: 2
```

Here is the algorithm:

```
Inside_Fill(Volume V)
   Inside_Fill_recurs(V,0)



Inside_Fill_recurs(Volume V,int v)
     if V.getValue(v) == 2      // if voxel v has  a value of 2
                                 // this means that we are still outside
         V.setValue(v,0)        // give it a value of 0 to label it an
                                 // outside voxel



     // look at 6 face neighbors



     if top_of_v exists            // top_of_v is the voxel directly above v
         Inside_Fill_recurs(V,top_of_v)
     if bottom_of_v exists         // bottom_of_v is the voxel directly below v
         Inside_Fill_recurs(V,bottom_of_v)
     if front_of_v exists          // front_of_v
         Inside_Fill_recurs(V,front_of_v)
     if back_of_v exists           // back_of_v
         Inside_Fill_recurs(V,back_of_v)
     if left_of_v exists
         Inside_Fill_recurs(V,left_of_v)  // left_of_v
     if right_of_v exists
         Inside_Fill_recurs(V,right_of_v) // right_of_v
```

Appendix C

Dijkstra's algorithm

This algorithm is the standard Dijkstra's algorithm described in [33] and explained in detail in 3.2.4.

```
shortest_paths( Graph g, Node start )
    initialise_single_source( g, start )
    Visited := { 0 }        // Make Visited list empty
    Q := Vertices( g )      // Put the vertices in a priority Q
    while not Empty(Q)
        u := ExtractCheapest( Q );
        AddNode( S, u ); /* Add u to S */
        for each vertex v in Adjacent( u )
            relax( u, v, w )

initialise_single_source( Graph g, Node start )
   for each vertex v in Vertices( g )
       g.d[v] := infinity    // d is the cumulative cost array
       g.p[v] := nil         // p is the parent array
   g.d[s] := 0;
```

```
relax( Node u, Node v, double w[][] )
    if d[v] > d[u] + w[u,v] then
       d[v] := d[u] + w[u,v]
       p[v] := u
```

Appendix D

Optimized version of Dijkstra for Centerline Extraction Step

This algorithm is an optimized version of Dijkstra's algorithm for the purpose of extracting a shortest path between two defined nodes in a graph. It takes as additional input the end node and stops as soon as the node visited is the end node since an optimal path from the source to the end is found at that point.

In this implementation, we only insert the neighbor in the priority list Q when one of its neighbors is being visited. The reason is that we did not store a Graph datastructure in memory for our algorithm since this would allocate an additional bytes of space. Once a node is inserted in the Q, we check that it is not inserted again by keeping a flag. The only node that is inserted during the initialization phase is the start node. From there, the rest of the graph nodes get put on the queue successively at each expansion phase

The lines with comments are changed from the original Dijkstra's algorithm described in Appendix C.

```
shortest_paths( Graph g, Node start, Node end )  //*** takes the end node as
                                                       input *** //
```

```
    initialise_single_source( g, start )
    Visited := { 0 }
    Q := {start}                //***  Only put start vertex in Q ***//
     while not Empty(Q) && continue
        u := ExtractCheapest( Q );
        if u == end             //** Stop once we have reached
            continue = 0;            our end node ***//
        AddNode( S, u );
        for each vertex v in Adjacent( u )
            relax( u, v, w )

initialise_single_source( Graph g, Node start )
   for each vertex v in Vertices( g )
       g.d[v] := infinity
       g.p[v] := nil
   g.d[start] := 0;
   continue = 1;



relax( Node u, Node v, double w[][] )
    if Q.hasMember(v)                  //** if v has already been added to
                                            Q, do as usual ***//
      if d[v] > d[u] + w[u,v] then
         d[v] := d[u] + w[u,v]
         p[v] := u
    else                               //** add v since it has not been added
                                            to Q yet *** //
       d[v] := d[u] + w[u,v]
       p[v] := u
```

```
Q.addNode{v}
```

Bibliography

[1] D. Gering, A. Nabavi, R. Kikinis, N. Hata, L. O'Donnell, E. Grimson, F. Jolesz, P. Black and W. Wells. An integrated visualization system for surgical planning and guidance using image fusion and an open mr. *J. Magn. Reson. Imaging*, 13:967–975, 2001.

[2] L. Ries, M. Eisner, C. Kosary, B. Hankey, B. Miller, L. Clegg, B. Edwards. *http://seer.cancer.gov/csr/*, SEER Cancer Statistics Review, 1973-1999, National Cancer Institute, Bethesda MD, 2002.

[3] Cancer. *http://www.nlm.nih.gov/medlineplus/ency/article/001289.htm*, Medline Plus Medical Encyclopedia, 2002.

[4] Biopsy for polyps. *http://www.nlm.nih.gov/medlineplus/ency/article/003415.htm*, Medline Plus Medical Encyclopedia, 2002.

[5] F. Jolesz, W. Lorensen, H. Shinmoto, H. Atsumi, S. Nakajima, P. Kavanaugh, P. Saiviroonporn, S. Seltzer, S. Silverman, M. Phillips, R. Kikinis. Interactive virtual endoscopy. *AJR*, 169:1229–1235, 1997.

[6] S.C. Chen, D.S. Lu, J.R. Hecht et *al.* Ct colonography: value of scanning in both the supine and prone positions. Am. J. Rad., 172:pp. 595–599., 1999.

[7] Lorensen *et al.* Marching cube: A high resolution 3-d surface reconstruction algorithm. *Computer Graphics*, 21(3):163–169, 1987.

[8] Mayo Clinic. Biomedical image resource. *http://www.mayo.edu/bir*.

[9] S. Bartling, T. Rodt, R. Gupta, B. Weber, D. Nain, R. Kikinis, A. Pfoh, H. Becker. Volumen-ct: Experimentelle untersuchung einer neuen ct technologie in der felsenbeindiagnostik. Deutschen Rntgenkongress 2002, 2002.

[10] U. Ecke, L. Klimek, W. Muller, R. Ziegler, W. Mann. Virtual reality: Preparation and execution of sinus surgery. *Comp Aid Surg*, 3:45–50, 1998.

[11] P. Rogalla, M. Werner-Rustner, A. Huitema, A. van Est, N. Meiri, B. Hamm. Virtual endoscopy of the small bowel: Phantom study and preliminary clinical results. *European Radiology*, 8:563–567, 1998.

[12] D. Vining, S. Aquino. Virtual bronchoscopy. *Clin Chest Med*, 20(4):725–30, 1999.

[13] C. Kay, D. Kulling, R. Hawes, J. Young,P. Cotton. Virtual endoscopy - comparison with colonoscopy in the detection of space-occupying lesions of the colon. *Endoscopy*, 32(3):226–32, 2000.

[14] R. Summers, P. Choyke, N. Patronas, E. Tucker, B. Wise, M. Busse, H. Jr Brewer, R. Shamburek. Mr virtual angioscopy of thoracic aortic atherosclerosis in homozygous familial hypercholesterolemia. *J Comput Assist Tomogr*, 25(3):371–7, 2001.

[15] Haker *et al.* Nondistorting flattening for virtual colonoscopy. *MICCAI 2000 proceedings*, 2000.

[16] A. Schreyer, J. Fielding, S. Warfield, J. Lee, K. Loughlin, H. Dumanli, F. Jolesz, R. Kikinis. Virtual cystoscopy: Color mapping of bladder wall thickness. *Invest Radiol*, 35(5):331–334, 2000.

[17] B. Lorensen W. Shroeder, K. Martin. *The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics.* Prentice Hall, New Jersey, 1997.

[18] *Tcl/Tk 8.3 Manual.* 2002.

[19] Hearn and Baker. *Computer Graphics.* Prentice Hall, New Jersey, 1997.

[20] Lennox Hoyte MD, Julia R Fielding, MD, Eboo Versi MD, PhD, Ron Kikinis MD. Mr based 3d reconstruction of the female pelvis: the case of complete prolapse. *Surgical Planning Lab, Brigham and Women's Hospital, Case of the Month, October 1999.*

[21] Freeflight software. *http://www.vec.wfubmc.edu/software/.*

[22] The Multidimensional Image Processing Lab at Penn State University. *http://cobb.ece.psu.edu/projects/quicksee/quicksee.htm.*

[23] M. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. *Visualization 2001, San Diego, CA.*

[24] T. Nakagohri, F. Jolesz, S. Okuda, T. Asano, T. Kenmochi, O. Kainuma, Y. Tokoro, H. Aoyama, W. Lorensen, R. Kikinis. Virtual pancreatoscopy. *Comp Aid Surg*, 3:264–268, 1998.

[25] M. Fried, V. Moharir, H. Shinmoto, A. Alyassin, W. Lorensen, L. Hsu, R. Kikinis. Virtual laryngoscopy. *Annals of Otology, Rhinology and Laryngology*, 108(3):221–226, 1998.

[26] H. Blum. *A transformation for extracting new descriptors of shape.* MIT Press, Cambridge, MA, 1967.

[27] The hamilton-jacobi skeleton. *ICCV 1999*, pages 828–834.

[28] D Nain, S Haker, R Kikinis, W Grimson. An interactive virtual endoscopy tool. *Satellite Workshop at the Fourth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'2001)*, 2001.

[29] S. Haker, S. Angenent, A. Tannenbaum, and R. Kikinis. Nondistorting flattening maps and the 3d visualization of colon ct images. IEEE Trans. on Medical Imaging, 19:pp. 665–670., 2000.

[30] O. Cuisenaire. Distance transforms: Fast algorithms and applications to medical image processing. Phd thesis, Universite Catholique de Louvain, Laboratoire de telecommunications et teledetection, October 1999.

[31] T. Deschamps. Curve and shape extraction with minimal path and level-sets techniques. applications to 3d medical imaging. Phd thesis, University Paris-9 Dauphine, Ceremade laboratory, December 2001.

[32] O. Cuisenaire and B. Macq. Fast euclidean distance transformations by propagation using multiple neighbourhoods. *Computer vision and Image understanding*, 76, (2):163–172, 1999.

[33] C.E. Leisterson T. H. Cormen and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, New York, 1998.

[34] D. Vining. Virtual endoscopy: Is it a reality? Radiology, 200:pp. 30–31., 1996.

[35] B. Acar, S. Napel, D.S. Paik, P. Li, J. Yee, C.F. Bealieu, R.B. Jeffrey. Registration of supine and prone ct colonography data: Method and evaluation. Radiological Society of North America 87th Scientific Sessions, 2001.

[36] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Co., New York, 1997.

[37] A. L. Ratan. Learning visual concepts for image classification. *Ph.D. Thesis, A.I. Lab, MIT*, 1999.