

Distance-Field Based Skeletons for Virtual Navigation

Ming Wan¹

The Boeing Company
P.O. Box 3707, M/C 7L-40, Seattle, WA 98124-2207

Frank Dachille² and Arie Kaufman²

Department of Computer Science
State University of New York at Stony Brook, Stony Brook, NY 11794-4400

Abstract

We present a generic method for rapid flight planning, virtual navigation and effective camera control in a volumetric environment. Directly derived from an accurate distance from boundary (DFB) field, our automatic path planning algorithm rapidly generates centered flight paths, a skeleton, in the navigable region of the virtual environment. Based on precomputed flight paths and the DFB field, our dual-mode physically based camera control model supports a smooth, safe, and sticking-free virtual navigation with six degrees of freedom. By using these techniques, combined with accelerated volume rendering, we have successfully developed a real-time virtual colonoscopy system on low-cost PCs and confirmed the high speed, high accuracy and robustness of our techniques on more than 40 patient datasets.

Keywords: Distance fields, path planning, centerline, camera control, virtual navigation, volumetric environment, physically based modeling, virtual colonoscopy.

1. Introduction

With the rapid increases in both scale and complexity of virtual environments, an efficient navigation capability becomes critical in virtual reality systems. It is a combination of efficient flight-path planning, intelligent camera control model, and fast rendering techniques.

Traditional navigation techniques focused primarily on a *planned navigation* mode, where a movie was computed by moving the camera automatically along a precomputed flight path from one end to the other and generating a sequence of navigation frames [10,17]. Centerline algorithms were often used as the flight path to give wide views at the object center. Navigation frames were rendered *off line*. Although such a planned navigation provided a quick overview of the virtual environment, it was rather limited due to lack of user interaction.

Real-time rendering techniques enable a more flexible *interactive navigation*. Earlier interactive navigation [8], called *free navigation*, required the user to control the camera movement at

every step to walk through the entire virtual environment. Without any guidance, such a manual navigation was both time consuming and challenging. To overcome this problem, Galyean [6] proposed a *guided navigation*, which provided some guidance for the camera and allowed the user to control it when desired. Hong et al. [11] proposed a more intelligent physically based “submarine” camera control model immersed within a potential field to prevent the camera from penetrating the object boundary during the guided navigation. However, like other potential field methods, their camera model suffered from the local minimum problem [13]. In addition, their camera could navigate only inside a 1D tubular region without any branch or more complicated structures.

In this paper, we present a generic system framework for virtual navigation in a volumetric environment, comprising a rapid flight-path planning, an effective camera control, and an interactive volume rendering. Our framework is based on an accurate *distance from boundary* (DFB) field, which contains the Euclidian distance from each voxel inside the 3D volumetric environment to the nearest object boundary. In fact, we consider DFB values only for those voxels within a *navigable region* inside the virtual environment, where we can navigate by manipulating the virtual camera. We assume that in our system there is a single-continuous navigable region inside the 3D environment. Otherwise, we should create a DFB field for each separate region.

We first present a simple but efficient automatic flight-path planning algorithm in Section 2. Directly derived from the accurate DFB field, our algorithm generates highly centered paths at a much higher speed than the fastest centerline algorithms published. It also generates a compact shape description of the virtual environment.

Our camera control model is described in Section 3, which combines the benefits of both planned and interactive navigation. We provide two navigation modes: automatic fly-through and interactive walk-through. A physically-based camera control model provides a realistic and convenient control during interactive navigation. A potential field based on the DFB field is employed for real-time collision detection and avoidance. To escape from a local minimum or simply expedite the exploration, the auto-navigation can be selected at any time to automatically fly along our precomputed central paths.

Compared to the “submarine” camera control model described in [11] that was specifically designed for navigation inside the essentially 1D tubular colon, our navigation technique is more general, convenient, and robust, with less computational expenses. For example, our technique is applicable to an arbitrary-shaped

¹ Email: ming.wan@boeing.com

² Email: {dachille | ari}@cs.sunysb.edu

3D navigation region which can be much more complicated than a 1D tube, and it eliminates the local minimum problem in the potential field.

We further employ a fast volume rendering technique (e.g., [16,25]) at each navigation frame. As a result, we have reached interactive rendering rates at about 10 Hz on a low-cost PC.

By using the above new techniques in our generic framework, we have successfully developed a concrete real-time virtual colonoscopy system. In Section 4, our primary experimental results from more than 40 actual patient data sets have shown the high accuracy, robustness and real-time performance of our techniques.

2. Flight-path Planning

2.1 Design Concepts

The following properties are desirable for a flight-path planning [1,11,17]:

- (1) To obtain a wide view of the virtual environment, the path should stay away from the surface.
- (2) The path should be a one-voxel-wide simple path without any 2D manifolds or self-intersection.
- (3) Any two adjacent voxels on the path should be *directly connected*, i.e., at most one, two, or three of their 3D coordinates differ by one, forming a 6-, 18-, or 26-connected path.
- (4) The path planning procedure should be fast and automatic, which frees the user from having to engage in the data processing.

There has been a great deal of research on flight-path planning based on the object *centerline* or a set of connected centerlines named *skeleton*. Most of the centerline extraction algorithms can be divided into three categories: manual extraction, distance mapping, and topological thinning.

Manual extraction [7,9], which requires the user to manually mark the center of each object region slice by slice, neither satisfies property 4 nor guarantees property 3. It may violate property 1, because a center point in a 2D slice may not lie along the medial axis in the 3D space.

Distance mapping [17] often employed Dijkstra's shortest path algorithm [4] to extract the centerline or flight path rapidly with full automation. It satisfied properties 2 through 4. Unfortunately, it does not satisfy property 1 because the shortest path tended to hug corners at high-curvature regions. Efforts have been made to push the shortest path towards the object center by post-remedy [27], which did not completely solve the problem, or distance function adjustment [1,23] based on other measurements, which are more computationally expensive than our solution.

In contrast, topological thinning [10,20] generated more centered paths, by peeling off a volumetric object layer by layer until there was only one central layer left. This technique satisfies properties 1 through 3 very well, but it does not satisfy property 4, due to the expensive iteration. Recently, Paik et al. [19] accelerated this technique by incorporating the above shortest path method into the parallel thinning procedure. However, the manual detection of the tip for each branches needs to be improved.

In this section, we present a simple solution for centerline and flight path generation, which satisfies all four properties. Our centerline algorithm is derived from a concise but precise centerline definition based on the DFB field. *We define the centerlines to be the minimum-cost paths spanning the DFB field inside a volume data set.* This definition has the following advantages. It precisely defines the centerline for distance mapping techniques by focusing on a centered path rather than a shortest path. More importantly, it suggests a provably fast and accurate solution.

Our centerline algorithm consists of two main steps: first, build a minimum-cost spanning tree in the DFB field; second, extract the colon centerline and its branches from the tree. It is worth pointing out that DFB values used in our algorithm are the exact real Euclidian distances, computed by a fast four-path algorithm proposed by Saito et al. [22]. It only took about 20 seconds to complete a connected colon region in a 512×512×361 CT volume (see Figure 1). Our centerline extraction algorithm based on such an accurate DFB value turns out to be more accurate than the accurate thinning algorithms that used the approximation conformation metrics [2] for boundary distance measurements. Meanwhile, it achieves a much higher speed than the fastest distance mapping algorithms because of its low computational complexity and high optimization.

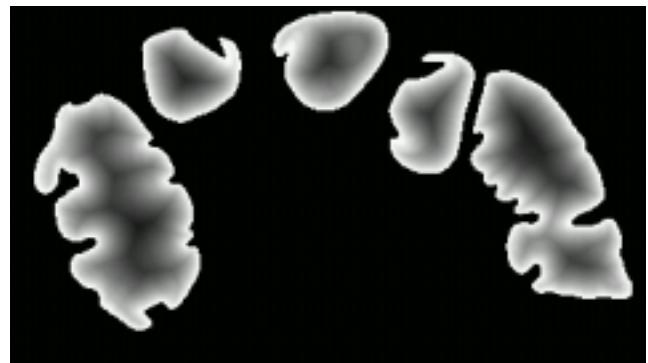


Figure 1. A cross-section of a 3D connected DFB field

2.2 Minimum-Cost Spanning Tree

The first step of our centerline algorithm can be separated into two stages. First, convert the volumetric DFB field into a 3D directed weighted graph. Second, build a minimum-cost spanning (MCS) tree from the weighted graph.

During the mapping from a volumetric data set to a 3D directed weighted graph, each voxel forms a node in the graph, while edges represent the 26-neighbor relations between voxels. Each edge has two directions pointing towards its two end nodes, respectively. Each direction has its own weight as the *inverse* of the DFB value of the end node it points to, in order to build a *minimum-cost* spanning tree. To distinguish the regular *DFB value* from its inverse, we call the latter *DFB cost*. Our mapping is different from those in the traditional distance mapping algorithms, where the length of each edge was used as its weight to form an undirected graph.

A minimum-cost spanning tree of our directed graph is defined as a tree that connects all the voxels in the navigable region at the minimum DFB cost. In order to build such a MCS tree one can

use the standard Dijkstra algorithm for computing the minimum cost of each node to reach a chosen source node S , where the source S is predefined by the user. Specifically, our algorithm computes the cost of a node at a voxel to be entered into the heap of discovered nodes as inverse of the distance from boundary at that voxel. We also accumulate the *distance from source* (DFS) as part of the Dijkstra algorithm progression according to: $dfs(B) = dfs(C) + distance(B, C)$.

In implementation, we adopt the fast heap-sorting technique to detect the node with the minimum DFB cost in the current heap in $O(\log N)$ time. Thereby, our algorithm is completed in $O(N \log N)$ time. In fact, the heap data structure is much simpler and faster than other sorting data structures that have the same computational complexity, such as the balanced binary tree.

As a result, the ordering of the voxels leaving the heap represents the connectivity of the MCS. Further, each inside voxel of the object contains a DFS value recording the length of this minimum DFB cost path. The DFS values will help us find the centerline and its branches in the subsequent step, and they will also contribute to interactive measurements during navigation when desired.

2.3 Centerline and Branch Extraction

This second step is critical for centerline and branch extraction. The centerlines extracted from the MCS tree give us a concise global picture of the topological features of the object, and they serve as a guide during navigation.

Our extraction procedure contains two stages: first, to extract a main centerline of the object ending at the source point S ; second, to detect all the branches attached to the main centerline. If the other end, say E , of the main centerline is given, then we can quickly find the sequence of voxels on the centerline by tracing back from E to S according to the MCS connectivity. Otherwise, our algorithm will automatically choose the voxel with the maximum DFS value as the other end of the centerline so that the centerline spans the entire object. From the perspective of our MCS tree, the centerline is the longest branch in the tree that starts from root S .

We implement an efficient branch extraction algorithm as follows, to automatically detect all the branches attached to the main centerline in $O(N)$ time.

Step 1. Scan centerline voxels by tracing back from the end E to the source S according to MCS connectivity (see Figure 2).

Step 2. For each centerline voxel C , check its $2d$ neighbors (excluding the two neighbors on the centerline) and find out each neighbor B that is MCS connected to C .

Step 3. For each B , find the MCS sub-tree rooted at B and identify voxel T with the largest DFS value among them. Then compute the length $len(B)$ between B and T as $dfs(T) - dfs(B)$. If $len(B)$ is larger than a user-specified threshold of the branch length, we store T as the tip of a branch growing from centerline voxel C through its neighbor B .

The above branch detection procedure is capable of detecting all branches directly connected to the centerline, including those

associated with the same centerline voxel. To further detect higher-level branches when desired, one can extend the above algorithm by recursively scanning each branch to find out all the higher-level branches, but the computational expenses increase.

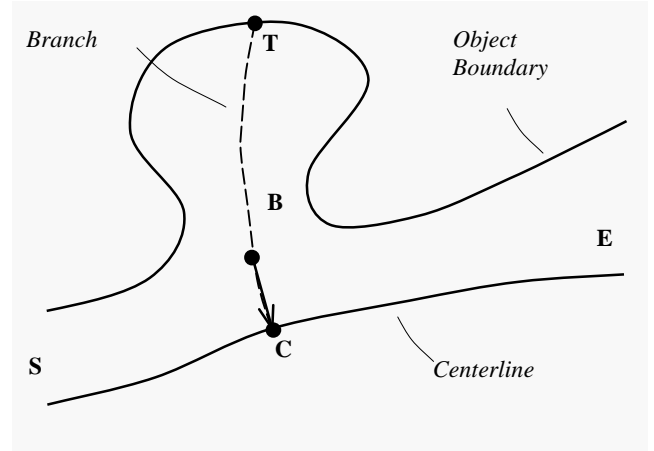


Figure 2. Branch detection at centerline voxel C .

From our experience, the main centerline gives the most compact shape description of the object, while the first-level skeleton consisting of the centerline and its directly attached branches provides sufficient shape information for navigation needs. Centerline with multiple level branches is only needed for very complex objects where a very detailed navigation map is required. By adjusting the threshold of branch length in our last recursive solution, we can easily control the level of detail during the branch extraction.

3. Camera Control Model

Our camera control model supports two navigation modes, interactive walk-through and automatic fly-through.

3.1 Interactive Walk-Through

In our interactive navigation mode, the user controls camera movement with six degrees of freedom using mouse buttons. We define three different mouse buttons for forward movement, backward movement, and no movement, respectively. At the same time, if the mouse cursor moves toward the left or right of the screen, we perform *yaw* rotation around the up direction of the camera to the left or right. Similarly, if the cursor moves toward the top or bottom of the screen, *pitch* rotation is used to look up or down. If the cursor stays at the center of the screen, the camera moves along the current viewing direction. Although we could also utilize *roll* rotation to complete a six degree-of-freedom manipulation, we have found this functionality is not as critical as others because it does not provide more information.

In order to provide a realistic and convenient control of camera movement in 3D free space and also prevent penetrating the object boundary, we develop a physically based camera control model to determine the position and orientation at each navigation step. Then, the up direction of the camera can be computed according to the orientation and the horizontal direction. We approximate the camera as a very tiny cube with both mass and moment of inertia unity.

We first consider the simple case of a pure translation, i.e., the camera moves along the current viewing direction, either forward or backward. The following equation of motion applies for the camera, which is under the influence of three simulated forces:

$$\dot{P}(t) = F_{user}(t) - kP(t) - \nabla D(X). \quad (1)$$

$\dot{P}(t)$ is the time derivative of the camera linear momentum $P(t)$ at time t ; $F_{user}(t)$ is the external force applied to the camera by the user; $kP(t)$ is a viscosity force to help control the speed, where k is the friction coefficient; and $\nabla D(X)$ is the gradient of the potential field $D(X)$ based on our DFB field at the current camera position X , to prevent the camera from penetrating the object boundary. Although the above equation looks similar to that described in [11], our definition and implementation of its components are different. Note that the last force in our equation applies only when the camera moves very close to the object boundary so as to it does not bother the user most of the time. Equation 1 allows translations not along the viewing direction, but we later constrain translation to lie along the new viewing direction.

When the user presses a mouse button, a constant force $F_{user}(t)$ is applied to the camera, leading to a constant acceleration. The duration of this force is determined by the duration of the button press. At the same time, the viscosity force $kP(t)$ pushes against the direction of movement and increases with the speed. It will asymptotically eliminate the acceleration from $F_{user}(t)$ with a time constant defined by the system so that the camera moves at a system-defined maximum velocity. Once the user releases the mouse button, the acceleration from $F_{user}(t)$ is removed so that the speed quickly drops down to zero due to the viscosity.

To avoid penetrating the object boundary, we generate a repulsive force similar to that defined in [11] from a potential field based on our DFB field:

$$D(X) = \begin{cases} C_s \left(\frac{\rho}{dfb(X)} - 1 \right)^2, & 0 < dfb(X) < \rho \\ 0, & otherwise \end{cases} \quad (2)$$

C_s is a coefficient controlling the strength of the repulsive force; ρ defines its maximum influence distance, which is set to 2 to 3 voxel lengths so as to minimize its influence and allow the camera to pass through most narrow openings. (In the extreme case of a local minimum of $D(X)$, the user needs to switch to the automatic mode to move forward, as discussed later in Section 3.2.) Central differences are employed to estimate $\nabla D(X)$.

In addition to the translation depicted by Equation 1, the yaw and pitch rotations of the camera can be approximately described as follows:

$$\dot{L}_{yaw}(t) = \tau_{yaw}(t) + k_{yaw} \omega_{yaw}(t) \quad (3)$$

$$\dot{L}_{pitch}(t) = \tau_{pitch}(t) + k_{pitch} \omega_{pitch}(t) \quad (4)$$

$\dot{L}_{yaw}(t)$ is the time derivative of the angular momentum $L_{yaw}(t)$; $\tau_{yaw}(t)$ is a constant torque from the user input. Its duration is determined by the duration of the button press;

$k_{yaw} \omega_{yaw}(t)$ is a viscosity torque to stop the rotation of the camera, where $\omega_{yaw}(t)$ is the angular velocity and k_{yaw} is the friction coefficient. The description for the pitch rotation is similar.

We present the viewing direction vector $V(t)$ by a yaw angle $\alpha(t)$ and a pitch angle $\beta(t)$ at a time t as follows:

$$V(t) = [\cos \beta(t) \cos \alpha(t), \cos \beta(t) \sin \alpha(t), \sin \beta(t)]^T. \quad (5)$$

Therefore, by doubly integrating Equations 1, 3 and 4, applying certain initial values with the Euler method, we obtain the location and orientation of the camera at every time step Δt :

$$\alpha(t + \Delta t) = \alpha(t) + L_{yaw}(t) \Delta t, \quad (6)$$

$$\beta(t + \Delta t) = \beta(t) + L_{pitch}(t) \Delta t, \quad (7)$$

$$X(t + \Delta t) = X(t) + \|P(t) \Delta t\| V(t + \Delta t). \quad (8)$$

3.2 Automatic Fly-Through

Based on our path planning, our camera model can automatically fly through the navigable region towards the source along the centerline paths. Our path planning guarantees the automatic fly-through stays far away from the object boundary with wide views. To provide smoother navigation along the precomputed voxel-based flight path and keep a stable orientation during the fly-through, we employ either of two strategies: looking at a position several voxels ahead along the path [10], or looking at the position where the path first disappears around a corner [19]. The advantage of the latter is allowing the user to see potentially interesting structures as they first come into view.

Our camera control model described in this section is different from existing navigation techniques. Our automatic fly-through technique provides a flight path that neither collides with the object boundary nor hugs corners. In other respects, our interactive walk-through technique is more general and effective than the camera model used in [11] due to the following considerations. First, as a general technique, it supports navigating in a more complicated navigable region than a 1D tube. Second, we use only one repulsive force rather than two forces, so that our computational expenses are lower. Without the attraction influence from the destination, we free the user from the continuous pressure of being dragged forward. Furthermore, our repulsive force does not bother the user unless the camera moves too close to the object boundary. In addition, our automatic fly-through mode speeds up the virtual exploration and eliminates the annoying local minimum problem.

4. Application and Experimental Results

As a generic framework, our virtual navigation system can be applied for a wide variety of applications. In particular, we have developed a real-time virtual colonoscopy system on ordinary PCs by using our new techniques. It provides a flexible real-time navigation inside a 3D virtual colon model acquired from a continuous sequence of 2D CT slices scanned from the human colon, aiming at detecting early-stage colon polyps. It exhibits the following features compared to the previous system [10,11]. First, the high speed of our centerline algorithm has dramatically reduced the preprocessing time from hours to minutes, resulting in a prompt diagnosis. Second, our dual-mode camera control model

provides intuitive, smooth, and sticking-free navigation inside the colon without user training. Third, by using fast direct volume rendering (e.g., [25]) rather than geometric rendering, our system generates more realistic colon surface images that are closer to the real optical ones. Finally, the use of accelerated volume rendering in our system makes it available on low-cost PCs rather than high-end graphics workstations.

Table 1. Results of our centerline algorithm.

Data sets	Colon 1	Colon 2	Colon 3	Colon 4
Slices	361	389	370	371
Inside voxels	3,146K	2,884K	2,322K	3,338K
Length	2,079	1,871	2,431	2,231
Time (sec.)	15.97	14.63	11.53	16.85

Our centerline or path-planning algorithm was implemented on an ordinary PC with a single Intel Pentium 700 MHz processor and 655 MB memory. We have validated the accuracy, speed, and robustness of our algorithm on 44 human colon data sets. Our algorithm has shown its robustness by correctly extracting in a short time all the centerlines, branches, and flight paths from all these data sets. Table 1 lists four colon data sets and Figure 3 shows a picture of one of them from our extensive experimental results. These four colons were among the most time-consuming ones because of their lengths. The resolution of each CT slice was 512×512.

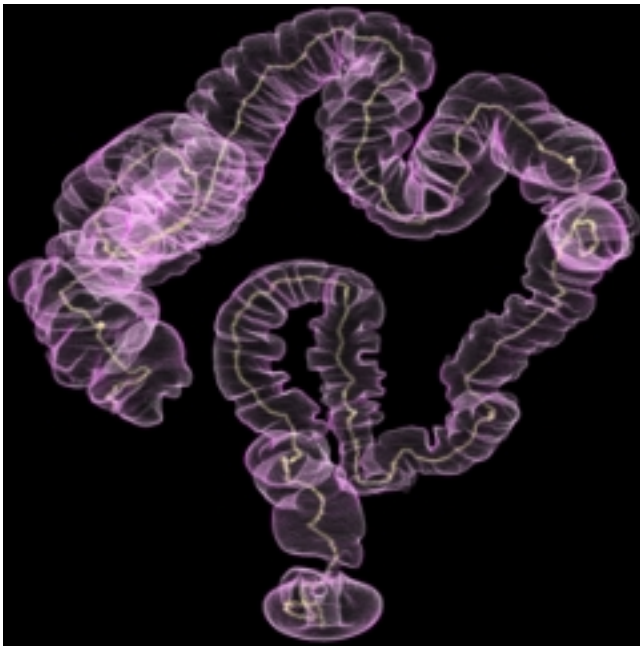


Figure 3. Centerline of colon 1 extracted in 16 second (see color plate in the color section).

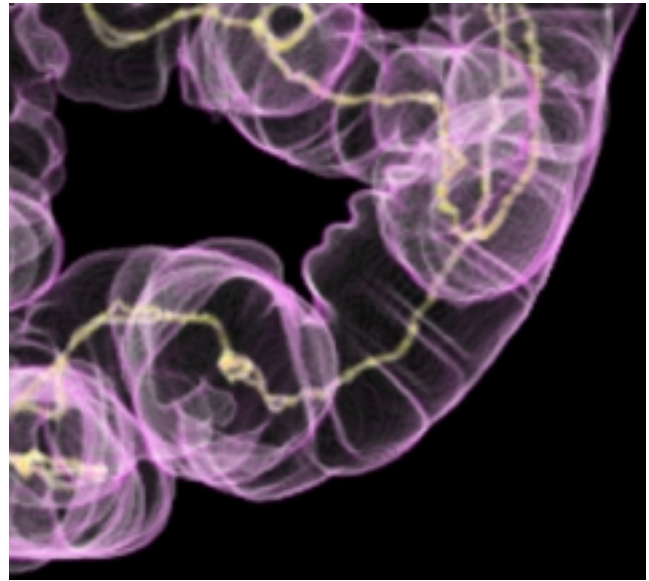


Figure 4. Two centerlines of colon 1 from different algorithms (see color plate in the color section).

Table 1 confirms the high speed of our centerline algorithm. It took about 12 to 17 seconds to complete path planning and extract the centerline. These times exclude the computation of the DFB and the times to setup the volumetric data structures enabling the fast computation of the DFS. In order to compare the performance of our algorithm with those of others, we ported and ran our algorithm on an SGI Power Challenge R10000 CPU with 4GB memory, the same machine used in [1,10,27]. We reached a similarly high performance as we found in Table 1. Therefore, our centerline algorithm turns out to be much faster than the fastest centerline algorithms published, which took 5 minutes (including DFB, DFS and centerline smoothing) [1] and 8 minutes [27] running on the SGI Power Challenge with the same colon data.

The centeredness of our centerlines and flight paths is theoretically derived from our accurate DFB field and visually confirmed by our virtual colonoscopy system. We further quantitatively verified it by comparing our centerline with a previous centerline that was generated by a topological thinning [10] and took more than 16 hours on the SGI Challenge. Figure 4 magnifies part of these two centerlines of colon 1. They were very close to each other and the average offset between them was 0.9 voxels wide. Two centerlines did not completely coincide due to the different measurements for boundary distances.

Since the human colon does not have an ideal topology to verify our branch detection algorithm, we used a 512×512×247 volume data set of a human rib cage in our experimentation. It took less than six seconds to detect one centerline and all 26 branches on our single-processor Intel Pentium 700, as shown in Figure 5. We found that the centerline of the cage was twisted rather than straight because of the irregular shape of the bones.

We successfully implemented our dual-mode camera control model during colon navigation. The interactive navigation mode was set as the default mode at the beginning of the navigation. The physician can press and hold a mouse button to navigate, and release the button for a stop. However, if the camera moves too

close to the colon wall, it bounces back due to the repulsive force. When the physician wants to auto-fly to speed up the navigation or escape from a local minimum in the potential field, he or she just double clicks the mouse button and release the mouse. The camera then switches to the auto-navigation mode and to fly to the other end of the colon. Another click quickly disengages auto-navigation and returns the control of the camera to the user.

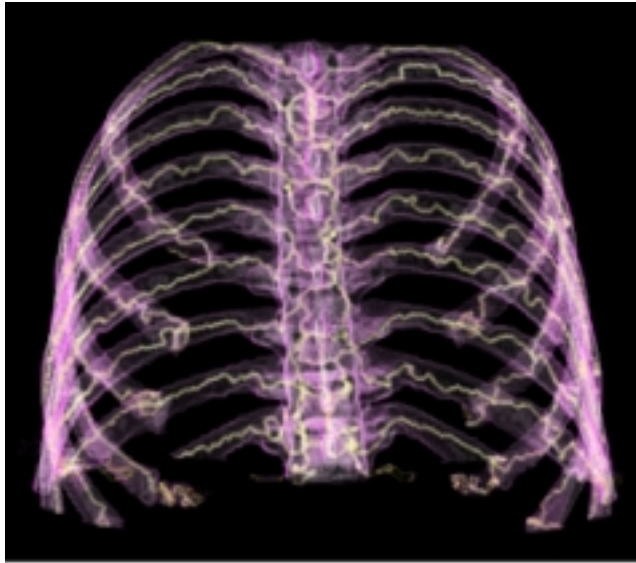


Figure 5. Centerline and its branches from a human rib cage (see color plate in the color section).

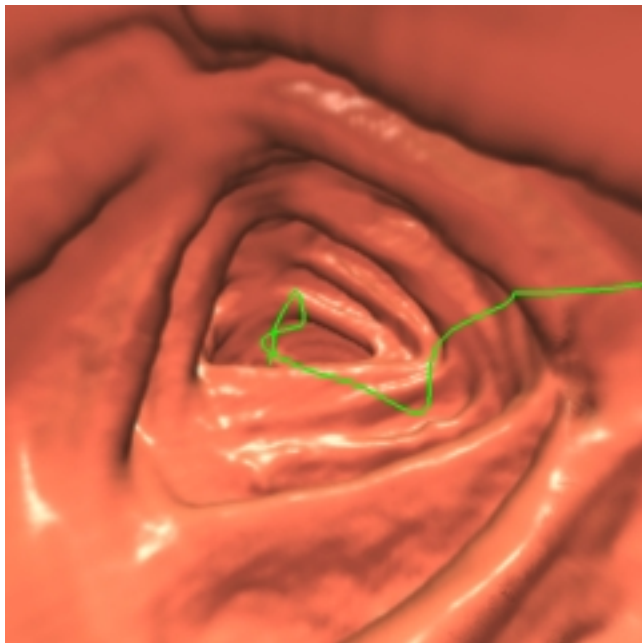


Figure 6. A navigation frame inside the human colon. The green line is the colon centerline (see color plate in the color section).

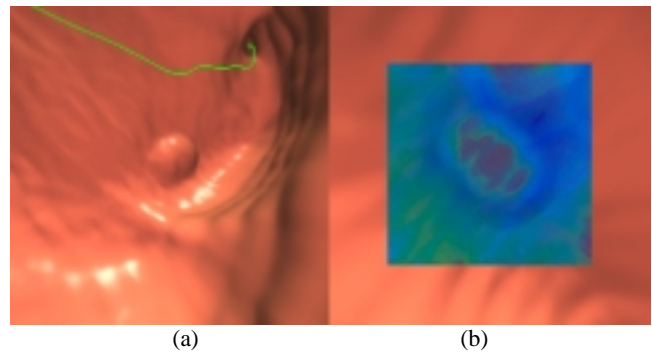


Figure 7. (a) A colon polyp detected in the interactive mode; (b) A translucent close view of the polyp interior for diagnoses (see color plate in the color section).

Figure 6 shows a photo-realistic colon image during navigation, rendered at an interactive speed of about 10 Hz frame rates on a fast PC with accelerated graphics (Nvidia GeForce 256, VolumePro board [21]). Higher frame rates in excess of 50 Hz can be attained at the expense of image quality. Compared to the geometric rendering results in [11], our volume-rendering image exhibited more natural and alias-free colon images, which was confirmed by physicians to be similar to what they observed in the optical colonoscopy.

Based on our effective and reliable navigation and visualization techniques, we have successfully detected all the colonic polyps that had been found in the traditional optical colonoscopy. Figure 7a shows an 8 mm polyp detected during our navigation, and Figure 7b displays an “electronic biopsy” view [26] of the polyp interior by using a more translucent transfer function during volume rendering.

5. Conclusions and Future Extension

We have presented a generic system framework for virtual navigation inside a complex volumetric scene. It provides a smooth, safe, and sticking-free navigation with immediate visual feedbacks with high image quality. It can apply to a wide range of applications, especially those high-fidelity applications in life science and industry. We demonstrated a real-time virtual colonoscopy system as a concrete application of our technique.

Although our framework is described to explore volumetric environments, it can be extended to deal with the traditional geometric object models by using voxelization techniques [24] to convert the polygon representation to volumetric models. For example, our centerline algorithm can be combined with McNeely et al’s voxel sampling technique [18] to create flight-path planning for effective haptic rendering in a complex industry CAD model consisting of millions of polygons.

Acknowledgements

This work has been supported by grants from NIH CA82402, ONR N000140110034, and Viatronix Inc. The patients’ data sets were provided by the University Hospital of the State University of New York at Stony Brook. We thank to Ingmar Bitter, Li Wei, Sarang Lakare, Kevin Kreeger, Mie Sato, Lichan Hong, Qi Ke, Jerome Liang, and Mark Wax for their contribution to the virtual colonoscopy project. Special thanks to William A. McNeely, Marie O. Murray, and Erin C. Allender for their helpful comments

on the drafts of this paper, and also to James J. Troy for his help on the video.

References

- [1] Bitter, I., M. Sato, M. Bender, K. McDonnel, A. Kaufman, and M. Wan, "CEASAR: A Smooth, Accurate and Robust Centerline Extraction Algorithm," *Proc. IEEE Visualization 2000*, 45-52.
- [2] Borgfors, G., "Distance Transformations on Digital Images," *Computer Vision Graphics Image Processing*, 1986, 34: 344-371.
- [3] Cabral, B., N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proc. Symposium on Volume Visualization '96*, 1996, 91-98.
- [4] Dijkstra, E., "A Note on Two Problems in Connexion the Graphs," *Numerische Mathematik*, 1959, 1: 269-271.
- [5] Durand, F., G. Drettakis, and C. Puech, "The Visibility Skeleton: A Powerful and Efficient Multi-Purpose Global Visibility Tool," *Proc. SIGGRAPH '97*, 1997, 89-100.
- [6] Galyean, T., "Guided Navigation of Virtual Environments," *ACM Symposium on Interactive 3D Graphics*, 1995, 103-104.
- [7] Ge, Y., D. Stelts, and D. Vining, "3D Skeleton for Virtual Colonoscopy," *Lecture notes in Computer Science*, 0302-9743m 1996, 449-454.
- [8] Gleicher M., and A. Witkin, "Through-the-Lens Camera Control," *Proc. SIGGRAPH '92*, 1992, 331-340.
- [9] Hara, A., C. Johnson, J. Reed, R. Ehman, and D. Ilstrup, "Colorectal polyp detection with CT Colography: Two- versus Three-Dimensional Techniques," *Radiology*, 1996, 200: 49-54.
- [10] Hong, L., A. Kaufman, Y. Wei, A. Viswambarn, M. Wax, and Z. Liang, "3D Virtual Colonoscopy," *Proc. Symposium on Biomedical Visualization*, 1995, 26-32.
- [11] Hong, L., S. Muraki, A. Kaufman, D. Bartz, and T. He, "Virtual Voyage: Interactive Navigation in the Human Colon," *Proc. SIGGRAPH '97*, 1997, 27-34.
- [12] Lacroute, P., "Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization," *Proc. Parallel Rendering Symposium*, 1995, 15-22.
- [13] Latombe, J., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
- [14] Levoy, M., "Display of Surfaces from Volume Data," *IEEE Computer Graphics & Application*, 1988, 8(5): 29-37.
- [15] Levoy, M., "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, 1990, 9(3): 245-261.
- [16] Li, W., A. Kaufman, K. Kreeger, "Real-Time Volume Rendering for Virtual Colonoscopy", *Volume Graphics Symposium 2001*, June, 2001.
- [17] Lorensen, W., F. Jolesz, and R. Kikinis, "The Exploration of Cross-Sectional Data with a Virtual Endoscope," in R. Satava and K. Morgan (eds.), *Interactive Technology and the New Medical Paradigm for Health Care*, 1995, 221-230.
- [18] McNeely, W., K. Puterbaugh, and J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling," *SIGGRAPH '99*, 1999, 401-408.
- [19] Paik, D., C. Beaulieu, R. Jeffery, G. Rubin, and S. Napel, "Automatic Flight Path Planning for Virtual Endoscopy," *Medical Physics*, 1998, 25(5): 629-637.
- [20] Pavlidis, T., "A Thinning Algorithm for Discrete Binary Images," *Computer Graphics and Image Processing*, 1980, 13: 142-157.
- [21] Pfister, H., J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, "The VolumePro Real-time Ray-Casting System," *Proc. SIGGRAPH '99*, 1999, 251-260.
- [22] Saito, T., and J. Toriwaki, "New Algorithm for Euclidean Distance Transformation of an N-Dimensional Digitized Picture with Applications," *Pattern Recognition*, 1994,27(11): 1551-1565.
- [23] Sato, M., and I. Bitter, and M. Bender, and A. Kaufman, and M. Nakajima, "TEASAR: Tree-structure Extraction Algorithm for Accurate and Robust Skeletons", *Pacific Graphics 2000*, 2000: 281—289.
- [24] Sramek, M., and A. Kaufman, "Alias-Free Voxelization of Geometric Objects," *IEEE Transactions on Visualization and Computer Graphics*, 5(3), 1999, 251-267.
- [25] Wan, M., W. Li, A. Kaufman, Z. Liang, D. Chen, and M. Wax, "3D Virtual Colonoscopy with Real-time Volume Rendering", in Physiology and Function from Multidimensional Images, Chin-Tu Chen, Anne V. Clough, Editors, *Proc. SPIE*, Vol. 3978, 2000, 165-170.
- [26] Wan, M., F. Dachille, K. Kreeger, S. Lakare, M. Sato, A. Kaufman, M. Wax, and J. Liang, "Interactive Electronic Biopsy for 3D Virtual Colonoscopy", *Proc. SPIE's International Symposium on Medical Imaging 2001*, San Diego, CA, February, 2001.
- [27] Zhou Y., and A. W. Toga, "Efficient Skeletonization of Volumetric Objects," *IEEE Transactions on Visualization and Computer Graphics*, 1999, 5(3): 196-209.