Interactive Dynamic Navigation for Virtual Endoscopy

Krishnan Ramaswamy[1,2] and William E. Higgins[2]

[1]Radionics Software Applications, Inc., Burlington, MA 01801

[2]Depts. of Electrical Engineering, Computer Science and Engineering, and the Bioengineering Program, Penn State University, University Park, PA 16802

Address of Corresponding Author:

William E. Higgins

Penn State University

121 Electrical Engineering East

University Park, PA 16802

fax: 814-863-5341

phone: 814-865-0186

Interactive Dynamic Navigation for Virtual Endoscopy

Krishnan Ramaswamy[1,2] and William E. Higgins[2]

[1]Radionics Software Applications, Inc., Burlington, MA 01801

[2]Depts. of Electrical Engineering, Computer Science and Engineering, and the Bioengineering Program, Penn State University, University Park, PA 16802

Abstract

Complex anatomical information can be obtained from a 3D radiologic image by navigating through it in a manner similar to an endoscopic examination. Real-time computation of "virtual" endoscopic views, however, is needed to permit interactive navigation. We present a fast volume-rendering method for computing such views. Our method, motivated by the temporal-coherence concept, performs dynamic volume rendering at interactive frame rates. Results demonstrate the method's efficiency and accuracy. Also, our method constitutes part of a complete virtual-endoscopic system we have devised. This system is illustrated for 3D pulmonary analysis.

Keywords: virtual endoscopy, endoscopy simulation, virtual bronchoscopy, interactive navigation, 3D radiologic imaging, pulmonary imaging, volume rendering.

1. INTRODUCTION

A high-resolution, 3D, radiologic image provides a virtual copy of the anatomy. Complex anatomical information can be obtained by interactive computer-based navigation through the image in a manner similar to an endoscopic examination. By the term *navigation*, we refer to the ability to travel inside the image in an arbitrary manner, computing local volume-rendered endoscopic views at each step [1]. Such dynamic navigation inside a 3D radiologic image has led to a new research area known as *virtual endoscopy* [1–17]. Virtual endoscopy generally implies traveling inside anatomical structures versus viewing 3D structures from their exterior [6, 8, 17]. The technique helps generate views similar to those seen in traditional endoscopy. Unlike traditional endoscopy, virtual endoscopy permits viewing of solid structures beyond hollow surfaces and permits navigation in any arbitrary direction.

Early virtual-endoscopic efforts point to a promising future, but they tend to suffer from the following restrictions: (1) they do not permit dynamic navigation at comfortable interactive speeds on an inexpensive computer; (2) they require substantial off-line analysis, either manual [11, 18] or semi-automatic [4], before they can be used for interactive dynamic navigation. Interactive speeds are, however, essential if the navigation is to be truly dynamic. This requires real-time volume rendering. *We present a fast volume-rendering method for virtual-endoscopic exploration on an inexpensive workstation.* The method forms part of a virtual-endoscopic system we have devised [9, 17]. We present algorithms and timing results for our method. We also illustrate our virtual-endoscopic system for 3D pulmonary analysis.

Volume visualization methods, in general, employ either surface rendering or volume rendering [19–22]. Surface rendering produces high-quality views from geometric polygonal representations of structures in the 3D image. Before surface rendering can be done, though, the image must be segmented and the segmentation must then be converted into a polygonal representation. Volume rendering, on the other hand, generates views directly from the 3D image. An ongoing debate exists regarding the merits of volume rendering versus surface rendering [13,17,21,22]. We have chosen volume rendering for our dynamic navigation method.

Much work has been carried out in developing fast volume-rendering methods. One approach has been to develop special-purpose hardware, but such hardware can be prohibitively expensive for routine use (e.g., [23, 24]). Other rendering methods attempt to cut down computation by using efficient algorithms. But they either sacrifice too much accuracy or do not permit navigation along arbitrary path trajectories [25–28]. For computing a dynamic sequence of views, researchers have suggested methods that exploit the *temporal coherence* between adjacent views [3, 28–34]. Our proposed volume-rendering method for navigation exploits the temporal-coherence concept.

The remainder of the paper is organized as follows. Section 2 sets up the navigation model and motivates the coherence concept. Section 3 discusses our proposed methods. Section 4 presents timing and accuracy results; it also illustrates the method's use in a complete virtual endoscopic system. Finally, Section 5 offers concluding remarks.

2. NAVIGATION GEOMETRY

Section 2.1 overviews the navigation scenario and describes the temporal-coherence concept. Section 2.2 sets up the geometry for ray casting in the navigation scenario.

2.1 Navigation Scenario and the Coherence Concept

Figure 1 depicts the general scenario for navigating through a 3D digital image. The user begins navigation at a selected *viewing site* $\mathbf{s}_1$ inside the image and then proceeds to trace out a *navigation path* $\mathbf{s}_1, \ldots, \mathbf{s}_m$ through the image. A volume-rendered virtual endoscopic view V_i is computed at each viewing site $\mathbf{s}_i$. V_i is a 2D image of the locally rendered data and is sometimes referred to as the *viewing screen.*

Interactive navigation requires that the rendered views be computed in near real-time. Unfortunately, the computational cost for brute-force ray-casting is high. For example, for a viewing screen of size $n \times n$ and rays of length n, brute-force ray casting takes $O(n^3)$ operations [20]. Fortunately, brute-force computations can be avoided if we realize that during navigation, the positional and directional changes from one viewing site to the next tend to be *incremental.* Hence, adjacent rendered views tend to have much similarity. This observation constitutes the temporal-coherence concept [3, 28–33].

For incremental viewing-site changes along a navigation path, the depths at which surfaces are encountered tend to change little from one viewing-site to the next (see Figure 2). We verified this observation experimentally using a $90 \times 90 \times 95$ 3D X-ray CT cardiac image highlighting the LV chamber. We computed the difference in depths at which surfaces are struck for two views situated at orientations $2°$ apart (we assumed that a surface was struck when a voxel along a cast ray exceeded a preset threshold). The observed depth differences are tabulated below:

85 % of the screen pixels changed ≤ 1 voxel in depth,

91 % changed ≤ 2 voxels in depth,

97 % changed ≤ 5 voxels in depth,

3 % changed > 5 voxels in depth.

The average ray changed 0.96 voxels in depth from one view to the next.

Hence, the depth information computed at viewing site $\mathbf{s}_i$ can conceivably be used to quickly obtain the rendered view at the next viewing site $\mathbf{s}_{i+1}$, without resorting to brute-force techniques. This observation forms the basis of our coherence method and helps to greatly reduce the computational cost during dynamic navigation. Either a parallel- or a perspective-viewing geometry can be assumed for the method. Also, the method is independent of the sampling technique used to compute samples along the rays. It uses depth cues in the scene to compute the volume-rendered views, and, thus, can be used in conjunction with shading techniques, such as depth shading, depth-gradient shading, and true-gradient shading [35, 36].

In our method, as shown later in Algorithm 1, a brute-force volume-rendered view V_1 is first computed at $\mathbf{s}_1$ (see Algorithm 2; all algorithms are presented in Section 3). Rendered views V_i at subsequent viewing sites $\mathbf{s}_2$, $\mathbf{s}_3$, ..., $\mathbf{s}_m$ are then computed by exploiting temporal coherence between consecutive views (see Algorithms 3-6); at each viewing site $\mathbf{s}_{i+1}$, the depth cues computed at the previous viewing site $\mathbf{s}_i$ are used to quickly obtain the volume-rendered endoscopic view V_{i+1}. Before describing these algorithms in Section 3, we first set up the basic ray-casting geometry.

2.2 Ray-Casting Geometry

Figure 3 illustrates the ray-casting model we use to compute the volume-rendered view at any viewing site $\mathbf{s}_i$. As shown in the figure, ray casting involves sending rays from each pixel on a $n \times n$ viewing screen V_i through the viewing pyramid. For our system, the viewing pyramid is contained within the

3D image (see Figure 1). Also, since our goal is to explore local subregions within the 3D image, the size of the viewing pyramid is determined by the desired field of view. Two coordinate systems are used in the ray-casting model: the object-coordinate system and the screen-coordinate system. All viewing relationships are initially specified in the screen-coordinate system and are subsequently transformed to the object-coordinate system using coordinate transformations. Before proceeding, we introduce some essential notation.

1. $\mathbf{s}_i = [s_x \quad s_y \quad s_z]$ denotes the i^{th} viewing site. It is a 3D location inside the image.
2. $\mathbf{d}_i = [d_x \quad d_y \quad d_z]$ is a unit viewing vector in the direction at $\mathbf{s}_i$.
3. $\mathbf{e}_i$ = eye location for $\mathbf{s}_i$.
4. V_i is the $n \times n$ viewing screen situated about $\mathbf{s}_i$.
5. $(j,\ k)$ = location of a pixel on V_i.
6. $\mathbf{w}_i = [w_x \quad w_y \quad w_z]$ is the unit vector along any horizontal row of the viewing screen.
7. $\mathbf{h}_i = [h_x \quad h_y \quad h_z]$ is the unit vector along any vertical column of the viewing screen.
8. Δw = distance between adjacent viewing-screen pixels in the direction of $\mathbf{w}_i$.
9. Δh = distance between adjacent viewing-screen pixels in the direction of $\mathbf{h}_i$. The viewing screen V_i is in general a rectangular array, so $\mathbf{w}_i \perp \mathbf{h}_i$.
10. $\vec{\rho}_i$ represents a ray cast during rendering.
    $\vec{\rho}_i(x, y, z, \cdot)$ is a specific ray emanating from point $[x \quad y \quad z]$ on the viewing screen.
    $\vec{\rho}_i(x, y, z, r)$ returns the coordinates of the original 3D image voxel struck by a ray that starts at $[x \quad y \quad z]$ and traverses a depth r samples along the ray.
11. r_{max} = maximum ray length, in number of samples.
12. Δr = actual distance between adjacent samples along a ray.

Bold face quantities represent 3D vectors or points and the subscript i refers to viewing site index. All screen-coordinate parameters are denoted by the superscript 's' to distinguish them from object-coordinate system parameters that have no superscripts. The viewing vector is aligned along the z^s axis of the screen-coordinate system, with the viewing screen in the $x^s - y^s$ plane at $z^s = 0$. The viewing site is situated at $(j, k) = (0, 0)$ of the viewing screen. The location and dimensions of the viewing pyramid for a given viewing site $\mathbf{s}_i$ are determined by $\mathbf{e}_i$, n, Δw, Δh, and r_{max}. The region of interest inside the viewing pyramid is assumed to be in positive z^s space with depth increasing in the positive z^s direction. The eye is located along the negative z^s axis at a distance of z_e^s units from the viewing screen. Thus, relative to the screen coordinate system, the current (i^{th}) viewing site and direction are

$$\mathbf{s}_i^s = [0 \ \ 0 \ \ 0], \quad \mathbf{d}_i^s = [0 \ \ 0 \ \ 1]$$

and the viewing pyramid is specified by

$$\mathbf{e}_i^s = [0 \ \ 0 \ \ -z_e^s], \quad \mathbf{w}_i^s = [1 \ \ 0 \ \ 0], \quad \mathbf{h}_i^s = [0 \ \ 1 \ \ 0]$$

for all viewing sites $i = 1, 2, \ldots, m$. Let $[x^s_{screen} \ \ y^s_{screen} \ \ 0]$ be the viewing-screen pixel at which a particular ray starts. The $[x^s \ \ y^s \ \ z^s]$ coordinates for samples along the ray can be obtained for $r = 0, 1, \ldots, r_{max} - 1$

as

$$x^s = x^s_{screen} + r\Delta r D^s_x, \quad y^s = y^s_{screen} + r\Delta r D^s_y, \quad z^s = r\Delta r D^s_z$$

$$D^s_x = \frac{x^s_{screen}}{\sqrt{(x^s_{screen})^2 + (y^s_{screen})^2 + (z^s_e)^2}}, \ D^s_y = \frac{y^s_{screen}}{\sqrt{(x^s_{screen})^2 + (y^s_{screen})^2 + (z^s_e)^2}}, \ D^s_z = \frac{z^s_e}{\sqrt{(x^s_{screen})^2 + (y^s_{screen})^2 + (z^s_e)^2}},$$

for a perspective-projection geometry; $D^s_x = 0$, $D^s_y = 0$, and $D^s_z = 1$ for a parallel-projection geometry. Thus, in a parallel-projection geometry, all viewing rays are parallel to each other and travel in the same direction as $\mathbf{d}^s_i$, the i^{th} viewing site's direction in screen coordinates. Before the rays can be cast, the above relations need to be defined in the object-coordinate system. Let (α, β) be the tilt and pan of the viewing screen in the screen-coordinate system relative to the object coordinate system. Any vector $[x^s \ \ y^s \ \ z^s]$ in the screen-coordinate system can be transformed to its counterpart $[x \ \ y \ \ z]$ in the object-coordinate system using

$$[x \ \ y \ \ z] \ = \ [x^s \ \ y^s \ \ z^s] \cdot R \ + \ T$$

$$R \ = \ R_\alpha \cdot R_\beta, \quad T \ = \ [s_x \ \ s_y \ \ s_z]$$

$$\text{where} \quad R_\alpha = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{pmatrix} \quad \text{and} \quad R_\beta = \begin{pmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\beta) & 0 & cos(\beta) \end{pmatrix}.$$

R accounts for the tilt and pan rotations and T accounts for the translation to the viewing site. Thus, $\mathbf{d}_i$, $\mathbf{w}_i$, $\mathbf{h}_i$, and $\mathbf{e}_i$ are defined in the object-coordinate system as

$$\mathbf{d}_i \ = [d_x \ \ d_y \ \ d_z] \ = \ [0 \ \ 0 \ \ 1] \cdot R, \qquad \mathbf{w}_i \ = [w_x \ \ w_y \ \ w_z] \ = \ [1 \ \ 0 \ \ 0] \cdot R \tag{1}$$

$$\mathbf{h}_i \ = [h_x \ \ h_y \ \ h_z] \ = \ [0 \ \ 1 \ \ 0] \cdot R, \qquad \mathbf{e}_i \ = [e_x \ \ e_y \ \ e_z] \ = \ [0 \ \ 0 \ \ -z^s_e] \cdot R + T \tag{2}$$

Using (1) and (2), rays can be cast for all viewing-screen pixels using a chain-coding algorithm [35]. For V_i, the 3D object coordinates of any viewing-screen pixel (j, k) are computed as

$$screenLoc = \ [x_{screen} \ y_{screen} \ z_{screen}] \ = \ \mathbf{s}_i \ + \ (k \cdot \Delta w)\mathbf{w}_i \ + \ (j \cdot \Delta h)\mathbf{h}_i \tag{3}$$

$$= \ [s_x \ s_y \ s_z] \ + \ (k \cdot \Delta w) \ [w_x \ w_y \ w_z] \ + \ (j \cdot \Delta h)[h_x \ h_y \ h_z] \tag{4}$$

where $\frac{-n}{2} \leq j, \ k < \frac{n}{2}$ and n is assumed to be even. The increment for stepping from one ray sample to the next is obtained as

$$rayInc = [x_{inc} \ y_{inc} \ z_{inc}] = [\Delta r \cdot D_x \quad \Delta r \cdot D_y \quad \Delta r \cdot D_z] \tag{5}$$

where $[D_x \ D_y \ D_z]$ are computed by transforming $[D^s_x \ D^s_y \ D^s_z]$ to the object-coordinate system.

To compute a rendered view V_i, we define the following operations and data structures:

1. $f(\vec{\rho}_i(x,y,z,r))$: gray-scale value of the nearest 3D image voxel struck by $\vec{\rho}_i(x,y,z,r)$. This may computed using nearest-neighbor or trilinear interpolation.

2. $g(f(\cdot))$: surface-test operator defined as

$$g(f(\cdot)) = \begin{cases} 1, & \text{if } f(\cdot) \text{ corresponds to a surface of interest} \\ 0, & \text{otherwise} \end{cases}$$

   An example surface-test criterion is one in which a ray sample $[x \quad y \quad z]$ is considered as belonging to a structure of interest if $f(\vec{\rho}_i(x,y,z,r))$ lies between preselected low and high thresholds; i.e., $v_{low} \leq f(\vec{\rho}_i(x,y,z,r)) \leq v_{high}$. A surface of interest is said to be struck by ray $\vec{\rho}_i(x,y,z,\cdot)$ if $g(f(\vec{\rho}_i(x,y,z,r)) = 1$ for some point $[x \ y \ z]$ along the ray.

3. $\rho_i^{type}[\ ]$: $n \times n$ array such that for screen pixel $(j,\ k)$

$$\rho_i^{type}[j,\ k] = \begin{cases} 1 \ \ (valid\_ray), & \text{if a structure of interest is struck} \\ 0 \ \ (empty\_ray), & \text{otherwise} \end{cases}$$

4. $\rho_i^{strike}[\ ]$: $n \times n$ array such that for screen pixel $(j,\ k)$

$$\rho_i^{strike}[j,\ k] = \begin{cases} r_{strike}, & \text{for a } valid\_ray \\ -\infty \ \ (NULL), & \text{for an } empty\_ray \text{ (no surface struck)} \end{cases}$$

   where r_{strike} is the depth of strike in number of ray samples; i.e., it is the first point r along the ray satisfying $g(f(\vec{\rho}_i(x,y,z,r)) = 1$.

Consider an arbitrary navigation trajectory $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$, ..., with respective viewing directions $\mathbf{d}_1$, $\mathbf{d}_2$, ... At $\mathbf{s}_{i+1}$, the navigation direction $\mathbf{d}_{i+1}$ in general changes from $\mathbf{d}_i$. Consecutive viewing sites are related by

$$\begin{aligned} \mathbf{s}_{i+1} &= \mathbf{s}_i + \mathbf{d}_{i+1} \cdot \Delta\mathbf{s} \\ \mathbf{d}_{i+1} &= \mathbf{d}_i + \Delta\mathbf{d}_{i+1} \end{aligned} \tag{6}$$

where $\Delta\mathbf{s}$ is the incremental distance moved from $\mathbf{s}_i$ to $\mathbf{s}_{i+1}$ and $\Delta\mathbf{d}$ is the incremental change in viewing direction from $\mathbf{s}_i$ to $\mathbf{s}_{i+1}$. In our method, we restrict $\Delta\mathbf{s}$ to be an integer multiple of the inter-sample distance Δr along a ray. Also, we consider both incremental forward and backward movements. Note that for a pure forward/backward translation, $\Delta\mathbf{d} = [0 \ \ 0 \ \ 0]$, and for a pure rotation, $\Delta\mathbf{s} = [0 \ \ 0 \ \ 0]$.

Now, consider an arbitrary viewing-screen location $(j,\ k)$ corresponding to a valid ray at $\mathbf{s}_i$ with a depth of strike $\rho_i^{strike}[j,k]$. Let the depth of strike for the corresponding $(j,\ k)$ location at $\mathbf{s}_{i+1}$ be $\rho_{i+1}^{strike}[j,k]$. Then, the change in depth of strike is given by

$$\Delta\rho^{strike}[j,k] = \rho_{i+1}^{strike}[j,k] - \rho_i^{strike}[j,k]$$

If $\Delta\mathbf{s}$ and $\Delta\mathbf{d}$ are small, then $\Delta\rho^{strike}[j,k]$ is also likely to be small, since the distance of surface points on the structure of interest has changed little from the previous site to the current site. This observation motivates the broad hypothesis driving our method:

For an incremental change in viewing site and direction from $(\mathbf{s}_i, \mathbf{d}_i)$ to $(\mathbf{s}_{i+1}, \mathbf{d}_{i+1})$, we can quickly determine the new depth of strike $\rho_{i+1}^{strike}[j,k]$ at a viewing-screen location (j,k) by performing a small search, beginning at $r = \rho_i^{strike}[j,k]$ along the corresponding ray $\vec{\rho}_{i+1}(x_{screen}, y_{screen}, z_{screen}, \cdot)$, per (4).

The main assumption in our method is that the changes in viewing position and direction are incremental; i.e. $\Delta\mathbf{s}$ and $\Delta\mathbf{d}$ are small. Let Δx, Δy, and Δz denote the sampling intervals of the original 3D image. Then, $\Delta\mathbf{s} \approx min(\Delta x,\ \Delta y,\ \Delta z)$ and $\Delta\mathbf{d} = \mathbf{d}_{i+1} - \mathbf{d}_i \leq \arctan(1/4)$ are sufficient guidelines (the suggestion for $\Delta\mathbf{d}$ implies a 0.25 unit shift from $\mathbf{d}_i$ to $\mathbf{d}_{i+1}$). If these constraints are relaxed, the method will not necessarily outperform brute-force rendering and will also tend to be inaccurate. Figure 2 illustrates the computational savings obtained for a single ray when performing an incremental movement. The figure shows rays cast from corresponding screen-pixel locations (j,k) at two adjacent viewing sites $\mathbf{s}_i$ and $\mathbf{s}_{i+1}$. At $\mathbf{s}_i$, the depth of strike at viewing-screen pixel (j,k) is $\rho_i^{strike}[j,k] = 7$ ray samples. Using brute-force ray-casting at $\mathbf{s}_{i+1}$, a distance of 6 ray samples has to be traveled before the surface is struck. If, instead, the coherence algorithm is used at $\mathbf{s}_{i+1}$, starting at depth $\rho_i^{strike}[j,k]$, a distance of one ray sample has to be traveled before the surface is struck. The next section details the algorithms for the coherence method.

3. ALGORITHMS

This section presents a series of six algorithms constituting the coherence-based dynamic volume-rendering method. Algorithm 1 gives a top-level synopsis of the method. Algorithms 2-6 are called within the body of Algorithm 1. To begin navigation at $\mathbf{s}_1$ (Algorithm 2), a brute-force calculation must first be done; initial depth information is recorded and the initial view V_1 is computed. For subsequent sites $\mathbf{s}_i$, $i = 2, 3, \ldots$, along a trajectory, the depth information is updated and new views are computed quickly. Algorithms 3 and 4 are general coherence algorithms for rendering views corresponding to any incremental forward or backward movement along a slowly changing navigation path. Algorithms 5 and 6 discuss improved algorithms for rendering views corresponding to pure translational movements made using a parallel-projection viewing geometry. The algorithms are straightforward to implement and give a general volume-rendering technique for interactive navigation.

Before proceeding, we define two operators used in Algorithms 2-5:

1. **forwardRaySearch(**$screenLoc$, $rayInc$, r_{start}, r_{end}, r_{strike}**)**: operator that steps along a ray in the forward direction from r_{start} to r_{end} ($r_{start} < r_{end}$). It returns TRUE if a structure of interest is struck and r_{strike} will contain the depth of the surface. It returns FALSE otherwise. The quantities $screenLoc$ and $rayInc$ are given by (4-5).

2. **backwardRaySearch(**$screenLoc$, $rayInc$, r_{end}, r_{start}, r_{strike}**)**: similar to operator **forwardRaySearch()**, but it instead steps along a ray in the backward direction from r_{end} to r_{start} ($r_{end} \geq r_{start}$).

Algorithms 2-5 use the arrays $\rho_1^{strike}[\,]$ and $\rho_1^{type}[\,]$ to compute the desired rendered views $V_i,\ i = 1, \ldots, n$. Depth shading, depth-gradient shading, or true-gradient shading can be used here [35]. For example, a depth-shaded rendered view using an eight-bit color scale can be computed as follows:

$$V_i(j,k) = \begin{cases} 0, & \text{if } \ \rho_1^{type}[j,k] = 0 \\ 255 \cdot (r_{max} - \rho_1^{strike}[j,k])/r_{max}, & \text{otherwise} \end{cases}$$

Algorithm 2 describes the steps in computing a brute-force rendered view at $\mathbf{s}_1$. For each screen pixel $(j,\ k)$ on the $n \times n$ viewing screen, a ray $\vec{\rho}$ is cast into the viewing pyramid according to the ray-casting geometry described in Section 2. Each ray is allowed to travel in the forward direction until either: (1) a structure of interest is struck at a depth $r_{strike} < r_{max}$; or (2) the maximum ray length r_{max} is reached without encountering any structure of interest.

Algorithm 3 is used to render views for an incremental forward movement or a pure rotation at any viewing site $\mathbf{s}_i$, $i = 2, 3, ..., m$. For screen pixels (j,k) corresponding to an empty ray at $\mathbf{s}_{i-1}$, the ray is searched in the forward direction from zero depth as in brute-force rendering. For every other screen pixel (j,k), we first search backward along the ray beginning at $\rho_{i-1}^{strike}[j,k]$. If the structure of interest is not struck during the backward search, we then search forward beginning at $\rho_{i-1}^{strike}[j,\ k] + 1$. Algorithm 3 searches backward first, because it assumes forward movements: when we move forward, we get closer to surfaces we can see. Note that this search technique is more accurate than an *alternating* ray-search technique in which the search begins at $\rho_{i-1}^{strike}[j,k]$ and alternately searches one sample forward and one sample backward along the ray.

Algorithm 4 gives the method used to obtain rendered views for an incremental backward movement at any viewing site $\mathbf{s}_i$, $i = 2, 3, \ldots, m$. For screen pixels (j,k) corresponding to an empty ray at $\mathbf{s}_{i-1}$, the ray is searched in the forward direction from zero depth. For every other screen pixel (j,k), the search along a ray proceeds in three stages. To find new surfaces that enter suddenly as we move backward, we first search forward along the ray, beginning at zero depth, for a distance of $r_{add} \cdot \Delta\mathbf{s}$, where r_{add} equals a small integer > 1 for perspective viewing and 1 for parallel viewing (note: $\Delta\mathbf{s} \neq 0$ in this case). This search checks for new surfaces that might have appeared between V_{i-1} and V_i. The factor r_{add} accounts for diverging rays in a perspective viewing geometry and can be chosen based on the size of the viewing screen $(n \times n)$, focal length z_e^s, and ray length r_{max}. For perspective viewing, the greatest divergence occurs for rays passing through the perimeter of the screen. The choice of r_{add} does not affect the rendered value; it's purpose is to speed up the search. A rule of thumb we have used for picking this parameter is $r_{add} = 0.05 \cdot r_{max} \cdot \arctan(\theta/2)$, where θ is the field of view (FOV) angle and $\theta/2 = \tan(0.5(n-1)/z_e^s)$. If no new structure of interest is struck in the above first search, we next look for surfaces that might have appeared directly in front of previously observed surfaces. To do this, we perform a backward search along the ray starting at depth $\rho_{i-1}^{strike}[j,k] + r_{add} \cdot (\frac{\Delta s}{\Delta r})$. The additive factor $r_{add} \cdot (\frac{\Delta s}{\Delta r})$ accounts for the backward movement by $\Delta\mathbf{s}$ from $\mathbf{s}_{i-1}$ to $\mathbf{s}_i$. Finally, if no structure was struck in the above search, we perform a forward search along the ray beginning at $\rho_{i-1}^{strike}[j,k] + r_{add} \cdot (\frac{\Delta s}{\Delta r}) + 1$; this search accounts for surfaces seen previously in V_{i-1}.

Note that Algorithms 3 and 4 are *almost exact* in contrast to *exact* brute-force rendering methods. At viewing site $\mathbf{s}_i$, for each screen pixel (j,k) corresponding to a valid ray at $\mathbf{s}_{i-1}$, we start the search along a ray at a depth determined by $\rho_{i-1}^{\text{strike}}[j,k]$ instead of zero depth. This can potentially cause small discrepancies in computed depths when compared to a brute-force rendering method. Such discrepancies could occur when: (i) new structures appear in the viewing pyramid at $\mathbf{s}_i$ very close to the viewing screen; or (ii) surface discontinuities result from the discrete nature of the 3D image. These discrepancies tend to be very small and virtually unnoticeable when the rendered views form part of a dynamic sequence. Also, it is well-known that useful information can be obtained from a dynamic sequence of rendered views, even if the individual views are slightly degraded [29, 37, 38]. If the user desires a perfect view at a particular viewing site, he can stop navigation and compute the desired exact view. Section 4 shows that the errors are very small and do not accumulate during navigation.

When using a parallel-projection geometry, more efficient algorithms are possible for performing pure forward/backward translations. Note that for all rays, the viewing direction at $\mathbf{s}_i$ does *not* change from the viewing direction at $\mathbf{s}_{i-1}$. Our algorithms exploits this observation. Algorithms 5 and 6 illustrate the methods for forward and backward translations at $\mathbf{s}_i$, $i = 2, 3, \ldots, m$.

Referring to Algorithm 5 (Algorithm 6 is similar), at $\mathbf{s}_i$, for screen pixels (j,k) which correspond to an empty ray at $\mathbf{s}_{i-1}$, a forward search is performed along the last $\Delta\mathbf{s}$ samples of the corresponding ray to check if a new structure has appeared. For every other screen pixel (j,k), if $\rho_{i-1}^{strike}[j,k] < (\Delta\mathbf{s}/\Delta r)$, we perform a forward search along the ray from zero depth since the previous depth of strike $\rho_{i-1}^{strike}[j,k]$ is no longer valid. If $\rho_{i-1}^{\mathrm{strike}}[j,k] \geq (\Delta\mathbf{s}/\Delta r)$, the new depth of strike, $\rho_i^{\mathrm{strike}}[j,k]$ is obtained by subtracting $\frac{\Delta s}{\Delta r}$ from $\rho_{i-1}^{strike}[j,\ k]$.

For Algorithms 3-6, a rendered view at a site $\mathbf{s}_i$, $i = 2,\ 3,\ ..,\ m$, can be computed in approximately $O(k \cdot n^2)$ time, where k equals the average incremental distance (in samples) searched for each cast ray. The factor k depends on the ray length r_{max}, focal length z_e^s, screen size $(n \times n)$, and the 3D image's content. Typically, $k << n$, which gives a rendering time much smaller than the $O(n^3)$ factor required by brute-force rendering. Our results of Section 4 bear out this observation. Algorithms 1-6 are presented below as pseudocode. The operators **backwardRaySearch(·)** and **forwardRaySearch(·)** also appear.

Algorithm 1: *Top-level algorithm for coherence-based navigation*
Initialization (see Algorithm 2):
    At $\mathbf{s}_1$
        for each screen pixel $(j,\ k)$
            mark $\rho_1^{type}[j,k]$ if a surface is struck
            record depth of strike $\rho_1^{strike}[j,k]$ for structure of interest
        compute view V_1 using arrays $\rho_1^{strike}[\,]$ and $\rho_1^{type}[\,]$.
Navigation (see Algorithms 3-6):
    At $\mathbf{s}_i$, $i = 2,\ 3,\ \ldots,\ m$
        input incremental movement
        for each screen pixel $(j,\ k)$
            compute $\rho_i^{type}[j,k]$ and new depth of strike $\rho_i^{strike}[j,k]$
                using previous depth information $\rho_{i-1}^{type}[j,k]$ and $\rho_{i-1}^{strike}[j,k]$
        compute view V_i using arrays $\rho_i^{strike}[\,]$ and $\rho_i^{type}[\,]$.

Algorithm 2: *Compute a brute-force rendered view at* $\mathbf{s}_1$
    for j = $-\frac{n}{2}$ **to** $\frac{n}{2} - 1$
        for k = $-\frac{n}{2}$ **to** $\frac{n}{2} - 1$
            $\rho_1^{type}[j,k] = empty\_ray$
            compute *screenLoc* and *rayInc* /* Use (4) and (5) */
            if forwardRaySearch($screenLoc,\ rayInc,\ 0,\ r_{max},\ r_{strike}$) = TRUE
                $\rho_1^{strike}[j,k] = r_{strike}$
                $\rho_1^{type}[j,k] = valid\_ray$
    compute V_1 using $\rho_1^{strike}[\,]$ and $\rho_1^{type}[\,]$

Algorithm 3: *General coherence method for an incremental forward movement or pure rotation.*

```
for j = -n/2 to n/2 - 1
  for k = -n/2 to n/2 - 1
    ρ_i^type[j, k] = empty_ray  /* Initialize */
    compute screenLoc and rayInc /* Use (4) and (5) */
    /* In tests below, search backward first;
       next, if necessary, search forward;
       finally, if necessary, start from zero depth if ρ_{i-1}^type[j, k] is an empty_ray. */
    if (ρ_{i-1}^type[j, k] = valid_ray)
       if backwardRaySearch(screenLoc, rayInc, ρ_{i-1}^strike[j, k], 0, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
       else if forwardRaySearch(screenLoc, rayInc, ρ_{i-1}^strike[j, k] + 1, r_max, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
    else if forwardRaySearch(screenLoc, rayInc, 0, r_max, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
compute V_i using ρ_i^strike[ ] and ρ_i^type[ ]
```

Algorithm 4: *General coherence method for an incremental backward movement.*

```
for j = -n/2 to n/2 - 1
  for k = -n/2 to n/2 - 1
    ρ_i^type[j, k] = empty_ray  /* Initialize */
    compute screenLoc and rayInc /* Use (4) and (5) */
    /* In the test below, for valid rays,
       search the first  r_add · Δs/Δr ray samples for new structures.
       If no new structure is found, search
          backward and then forward from ρ_{i-1}^strike[j, k] + r_add · (Δs/Δr).
       If ρ_{i-1}^type[j, k] is an empty ray, start from zero depth. */
    if (ρ_{i-1}^type[j, k] = valid_ray)
       if forwardRaySearch(screenLoc, rayInc, 0, r_add · Δs/Δr, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
       else if backwardRaySearch(screenLoc, rayInc,
                          ρ_{i-1}^strike[j, k] + r_add · Δs/Δr, r_add · Δs/Δr, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
       else if forwardRaySearch(screenLoc, rayInc,
                          ρ_{i-1}^strike[j, k] + r_add · Δs/Δr + 1, r_max, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
    else if forwardRaySearch(screenLoc, rayInc, 0, r_max, r_strike) = TRUE
          ρ_i^strike[j, k] = r_strike
          ρ_i^type[j, k] = valid_ray
compute V_i using ρ_i^strike[ ] and ρ_i^type[ ]
```

Algorithm 5: *Pure forward translation for a parallel-projection geometry*

/* All rays have same viewing direction */
$x_{inc} = D_x \cdot \Delta r$
$y_{inc} = D_y \cdot \Delta r$
$z_{inc} = D_z \cdot \Delta r$
$rayInc = [x_{inc}\ y_{inc}\ z_{inc}]$
for j $= -\frac{n}{2}$ **to** $\frac{n}{2} - 1$
  for k $= -\frac{n}{2}$ **to** $\frac{n}{2} - 1$
    $\rho_i^{type}[j,\ k] = empty\_ray$
    compute $screenLoc$ /* Use (4) */
    if $(\rho_{i-1}^{type}[j,\ k] = valid\_ray)$
      if $(\rho_{i-1}^{strike}[j,\ k] < \frac{\Delta s}{\Delta r})$ /* Previous depth of strike
        at $\mathbf{s}_{i-1}$ is no longer valid, since we have moved past that depth */
        if forwardRaySearch$(screenLoc,\ rayInc,\ 0,\ r_{max},\ r_{strike})$ = TRUE
          $\rho_i^{strike}[j,\ k] = r_{strike}$
          $\rho_i^{type}[j,\ k] = valid\_ray$
      else /* New depth of strike can be directly computed */
          $\rho_i^{strike}[j,\ k] = \rho_{i-1}^{strike}[j,\ k] - \frac{\Delta s}{\Delta r}$
          $\rho_i^{type}[j,\ k] = valid\_ray$
    else /* $\rho_{i-1}^{type}[j,\ k]$ was empty. Search the last $\frac{\Delta s}{\Delta r}$ ray samples */
      if forwardRaySearch$(screenLoc,\ rayInc,\ r_{max} - \frac{\Delta s}{\Delta r},\ r_{max},\ r_{strike})$ = TRUE
          $\rho_i^{strike}[j,\ k] = r_{strike}$
          $\rho_i^{type}[j,\ k] = valid\_ray$
compute V_i using $\rho_i^{strike}[\ ]$ and $\rho_i^{type}[\ ]$

Algorithm 6: *Pure backward translation for a parallel-projection geometry*

/* All rays have same viewing direction */
$x_{inc} = D_x \cdot \Delta r$
$y_{inc} = D_y \cdot \Delta r$
$z_{inc} = D_z \cdot \Delta r$
$rayInc = [x_{inc}\ y_{inc}\ z_{inc}]$
for j $= -\frac{n}{2}$ **to** $\frac{n}{2} - 1$
  for k $= -\frac{n}{2}$ **to** $\frac{n}{2} - 1$
    $\rho_i^{type}[j,\ k] = empty\_ray$
    compute $screenLoc$ /* Use equation (4) */
    if $(\rho_{i-1}^{type}[j,\ k] = valid\_ray)$
      /* Search the first $\frac{\Delta s}{\Delta r}$ ray-samples for new structures */
      if forwardRaySearch$(screenLoc,\ rayInc,\ 0,\ \frac{\Delta s}{\Delta r},\ r_{strike})$ = TRUE)
        $\rho_i^{strike}[j, k] = r_{strike}$
        $\rho_i^{type}[j, k] = valid\_ray$
      else if $((\rho_{i-1}^{strike}[j, k] < (r_{max} - \frac{\Delta s}{\Delta r}))$ /* Surface point inside viewing pyramid */
        $\rho_i^{strike}[j, k] = \rho_{i-1}^{strike}[j, k] + \frac{\Delta s}{\Delta r}$
        $\rho_i^{type}[j, k] = valid\_ray$
    else /* $\rho_{i-1}^{type}[j, k]$ was an $empty\_ray$ */
      /* Search the first $\frac{\Delta s}{\Delta r}$ ray samples for new structures */
      if forward_ray_search$(screenLoc,\ rayInc,\ 0,\ \frac{\Delta s}{\Delta r},\ r_{strike})$ = TRUE
        $\rho_i^{strike}[j, k] = r_{strike}$
        $\rho_i^{type}[j, k] = valid\_ray$
compute V_i using $\rho_i^{strike}[\ ]$ and $\rho_i^{type}[\ ]$

backwardRaySearch($screenLoc,\ rayInc,\ r_{end},\ r_{start},\ r_{strike}$)

```
    /* screenLoc = [X_screen Y_screen Z_screen],  rayInc = [X_inc Y_inc Z_inc] */
    x_cur = X_screen + r_start · X_inc
    y_cur = Y_screen + r_start · Y_inc
    z_cur = Z_screen + r_start · Z_inc
    r = r_end
    while (r ≥ r_start)
        rayVal1 = f(ρ⃗_i(x_cur, y_cur, z_cur, r))
        /*  Fetch the (r − 1)^th sample along the ray.
          For the r^th ray-sample to be a surface point,
          the (r − 1^th) sample should fail the surface test.  */
        x_prev = x_cur - X_inc
        y_prev = y_cur - Y_inc
        z_prev = z_cur - Z_inc
        rayVal2 = f(ρ⃗_i(x_prev, y_prev, z_prev, r))
        if(     (g(rayVal1) = 1  AND  r = r_start)
           OR (g(rayVal1) = 1  AND  g(rayVal2) = 0) )
            r_strike = r
            return(TRUE)   /* Surface of structure found  */
        else
            x_cur = x_cur - X_inc
            y_cur = y_cur - Y_inc
            z_cur = z_cur - Z_inc
            r = r - 1
    return(FALSE) /* No structure of interest was struck */
```

forwardRaySearch($screenLoc,\ rayInc,\ r_{start},\ r_{end},\ r_{strike}$)

```
    /* screenLoc = [X_screen Y_screen Z_screen],  rayInc = [X_inc Y_inc Z_inc] */
    x_cur = X_screen + r_start · X_inc
    y_cur = Y_screen + r_start · Y_inc
    z_cur = Z_screen + r_start · Z_inc
    r = r_start
    while (r < r_end)
        rayVal=f(ρ⃗_i(x_cur, y_cur, z_cur, r))
        if (g(rayVal) = 1)
            r_strike = r
            return(TRUE) /*  Surface of structure found  */
        else
            x_cur = x_cur + X_inc
            y_cur = y_cur + Y_inc
            z_cur = z_cur + Z_inc
            r = r + 1
    return(FALSE) /* No structure of interest was struck */
```

4. RESULTS

We first present timing and accuracy results for the various rendering methods. Next, we illustrate the use of the coherence method for interactive navigation in a complete virtual-endoscopic system.

4.1 Numerical Results

All timing results were obtained on a Sun SPARCstation 10 workstation with four 50 Mhz processors. We used a $96 \times 96 \times 96$ phantom image in which all voxels are 0, except on slice $z = 95$, which is set to 255. All results were obtained using a 64×64 screen ($n = 64$) and a maximum ray depth of 64 ($r_{max} = 64$). This known phantom image allowed us to make known numerical determinations on algorithm performance.

Tables 1–2 give timing results for performing pure forward translations using perspective and parallel viewing geometries. We used Algorithms 3 and 5 to obtain the results. The timing results for incremental backward translations using Algorithms 4 and 6 are similar [9]. To obtain the tabulated results, we computed a sequence of 20 consecutive transverse-plane views for each of four different starting viewing sites $\mathbf{s}_1$. That is, we computed the sequence of 20 views $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_{20}$ for starting points $z = 40$, $z = 50$, $z = 60$, and $z = 70$. To get the 20 views starting at $z = 40$, views V_1, V_2, etc., were computed for $z = 40$, $z = 41$, $z = 42$, etc; i.e., we walk toward the plane of $255's$ at $z = 95$. Thus, the average ray depth encountered by the 20 views for starting point $z = 40$ is 45.5.

Tables 3-4 show timing results for pure rotations using Algorithm 3. We computed views corresponding to a sequence of 10 consecutive rotations, at intervals of 2°, for four different viewing sites, $z = 40$, $z = 50$, $z = 60$, and $z = 70$. Note that $\mathbf{s}_i$ does not change in this case; only $\mathbf{d}_i$ changes. To get the 10 views starting at $z = 40$, views V_1, V_2, V_3, etc., were computed for $\theta = 0°$, $\theta = 2°$, $\theta = 4°$, etc., where θ is the angle of rotation about the (vertical) y axis of the image. Rays were cast using nearest-neighbor sampling.

For Tables 1-4, T_1 equals the time required for rendering alone, and T_2 is the total time required for rendering, shading, and display. Thus, $T_2 > T_1$. $T_2 - T_1$, the time required to put the final image on the display console, is independent of the rendering method. Also, the viewing screen $= 64 \times 64$, the maximum ray depth $r_{max} = 64$, the focal length for perspective viewing $z_e^s = 30$. These parameters correspond to a field of view (FOV) of 93^o, similar to that of a typical endoscope, which has an FOV of 100^o. All times measure the average time taken per movement in seconds.

The timing results for this example show that the coherence method greatly outperforms brute-force ray casting in all cases, typically on the order of a factor of 5-10. The greatest speed gain over brute-force rendering is achieved when performing pure translations using a parallel-projection geometry. For the parallel geometry, the coherence method's rendering time T_1 rivals $T_2 - T_1$, the time needed to complete the display of the image. Translating the times into frame rates, the coherence method gave results ranging from 8 frames/sec to 52 frames/sec. Note that 64×64 frames were computed. This follows the general concept of virtual-endoscopic navigation per Figure 1, where you only view what is within your local field of view. With current systems running at 300 Mhz and higher, these rates can be much higher and clearly in the realm of comfortable real-time interaction.

Algorithms 5 and 6 are exact rendering methods for performing pure translational movements using a parallel viewing geometry. But, as stated in Section 3, Algorithms 3 and 4 give rise to small discrepancies in the rendered views in comparison to brute-force ray casting. These discrepancies occur when (i) performing translational movements with a perspective viewing geometry, and (ii) performing rotational movements with either viewing geometry. Tables 5-6 show the average difference in surface-depths of strike between brute-force rendering and the coherence method. To generate the results, we used a 3D CT image of a canine thorax, generated using an Imatron EBCT scanner. The original data consisted of forty $3mm$-thick slices gathered over a single 52-sec breath hold; this data was interpolated to 0.703 mm z resolution. An 18kg dog was anesthetized and mechanically ventilated during the scan (functional residual capacity—0 cm H_20 lung pressure). The interpolated 3D image size is $256 \times 256 \times 155$, with voxels of size $0.703mm \times 0.703mm \times 0.703mm$. The original image's gray scale was inverted for clarity.

To obtain the results shown in Table 5, 100 consecutive translations were performed with a perspective viewing geometry using brute-force rendering and using Algorithms 3-4 of the coherence method. The results were computed after every 10 translations. The results of Table 6 were similarly obtained for both parallel and perspective viewing geometries by using Algorithm 3 to perform 100 consecutive viewing-site rotations. For Tables 5-6, the viewing screen = 64×64, r_{max} = 64, focal length for perspective viewing z_e^s = 30, and $r_{add} = 3$ (FOV = 93^o).

The tables show the average depth error per screen pixel, in terms of number of ray samples. Specifically, e_{nn} = average depth error per screen pixel [in terms of number of ray samples] for nearest-neighbor sampling and e_t = average depth error per screen pixel [in terms of number of ray samples] for trilinear-interpolation sampling. Note that these errors are very low (less than 1 ray sample) in all cases. Furthermore, in all cases, the errors do not monotonically increase with the number of movements; thus, errors do not accumulate as one moves. Figure 4 shows a sample comparison of the rendered views computed using the two methods. The views correspond to the results for trilinear-interpolation sampling with a parallel viewing geometry, as shown in the third row of Table 6. Note that much similarity appears in the views, even in this case where the average error in depth is relatively higher than most other cases shown in Tables 5 – 6. Also, note that the view computed using the coherence method is not a static view. It belongs to a dynamic sequence of views, and, hence, the visual discrepancies in views rendered using brute-force and coherence methods become much less discernible when performing interactive navigation. If desired, the discrepancies between views computed using the coherence method and a brute-force rendering method can be further reduced by: (1) using brute-force ray casting for a small set of randomly chosen screen pixels at each viewing site, or (2) using brute-force ray casting to render an entire view at every l^{th} viewing site, where l is a user-selected parameter.

4.2 Virtual-Endoscopic System

We have devised a computer-based system for virtual endoscopic exploration [9, 17]. The system was developed using C++ within the Solaris 2.X operating environment. Before invoking the system, a 3D image of the anatomical region of interest must be generated using a radiologic imaging scanner. If desired, off-line automatic processing can be performed first on the 3D image to generate navigation paths through the branching structures of interest [9, 39, 40]. Such automatic processing facilitates easy, detailed, interactive study of the anatomical structures later. It also alleviates much of the burden of creating navigation paths manually, especially for complex branching structures. To perform interactive navigation, the 3D image and any precomputed path trajectories are loaded into the system. During interactive analysis, the user can view, create, and edit navigation paths of interest. The system has many tools that can be used during interactive navigation [9,17]. The primary tool is the *Endoscope* that displays volume-rendered virtual-endoscopic views, computed using the coherence method. Other navigation tools also exist:

1. *Projections:* Provides global volume-rendered *projection* information.
2. *2D Slicer:* Provides global 2D orthographic slice information about viewing sites.
3. *Cross-Sections:* Gives small local cross-sections about viewing sites along a navigation path.
4. *Measurement:* Gives quantitative information of structures of interest.
5. *Tube Viewer:* Gives a mathematically straightened global view of image data along a path.
6. *Navigation Helm:* Acts as an interactive console for navigating manually through an image.

When viewing information along a path, information is presented interactively in all invoked tools simultaneously. This gives an interactive "movie" of information along a navigation path.

As an example of interactive exploration, we again consider the 3D CT image of the canine thorax. We computed various path trajectories through the pulmonary airways using off-line automatic processing [9]. After the paths were generated, they were loaded into the system for interactive exploration. Figures 5-6 illustrate the use of various tools in performing interactive exploration. Figure 5 shows selected frames of an endoscopic movie along a branch of one of the pulmonary airways. The *Projections* tool contains maximum-intensity projections of the 3D image in the coronal direction, on which the precomputed path trajectories are superimposed. The selected branch is darkened in the figure. The *Endoscope* shows rendered views from selected viewing sites along the branch. (In reality, an endoscopic movie is displayed to the user.) The volume-rendered views shown in the *Endoscope* are computed using the coherence method in Algorithm 3.

Figure 6 shows the *Endoscope*, *2D Slicer*, and *Cross-Sections* tools at a site along the branch selected in Figure 5. The *Endoscope* show a static 3D rendered view at the site. The *2D Slicer* provides global image information at the site. Axial, Coronal, and sagittal data planes passing through the current viewing site are shown in the *2D Slicer*. The *Cross-Sections* tool shows three mutually orthogonal 2D-slice images containing original image data passing through the current viewing site. These 2D-slice images, labeled as *Ortho* (orthogonal plane), *Vert* (vertical plane), and *Horiz* (horizontal plane), are computed relative to the current viewing direction (unlike the *2D Slicer*).

For the final example, Figure 7 demonstrates the proposed navigation methodology and system for a human case. In the example, a healthy human volunteer held his breath three separate times for twenty seconds each time [17]. During these breathholds, high-resolution EBCT scans were made of the top, middle, and bottom portions of the chest. These scans were assembled into one complete chest scan to give a 3D data set consisting of 116 slices, with 512×512 voxels per slice. Each slice had a thickness of 3.0mm and an axial-plane resolution of 0.781mm. This example shows the possibility of exploring specific sites on a navigation path (Figure 7c-d) and the option of manually navigating through the data (Figure 7e). References [9, 17, 40] give provide further examples using human data and more detailed information on the system.

5. DISCUSSION

We described a fast volume-rendering method for interactive navigation in a 3D image. The method is applicable to any scenario requiring volume-rendered views along an incrementally changing navigation path. The method is easily implemented on an inexpensive computer. It is particularly well-suited to virtual-endoscopic navigation, where the field of view is small compared to the entire size of the given 3D image. (The 3D image is the entire "virtual environment.") We presented timing results illustrating the superior performance of the method in comparison to navigation using brute-force volume rendering. The coherence method can be used with either a parallel or a perspective geometry. For virtual endoscopy, the perspective geometry more closely resembles the views of a real endoscope [17], but researchers have found it necessary to correct for the inherent geometric distortion [10,41]. A parallel-projection viewing geometry affords undistorted structures and greater potential for graphical measurement [17]. As a final comment, our method is well suited for interactive exploration within a complete virtual-endoscopic system. We illustrated this for 3D canine and human pulmonary cases. References [17, 39, 40] give further examples.

ACKNOWLEDGEMENTS

This work was partially supported by grants from the Whitaker Foundation and by grant #CA74325 from NIH NCI. The 3D image data were provided by Drs. Eric Hoffman and Geoffrey McLennan of the University of Iowa. Preliminary versions of this work were presented in the non-refereed conferences [3,42].

References

[1] M. L. Brady, W. E. Higgins, K. Ramaswamy, and R. Srinivasan, "Interactive navigation inside 3D radiological images," *Proc. Biomedical Visualization '95*, vol. Atlanta, GA, pp. 33–40 and 85, Oct. 1995.

[2] D. J. Vining, R. Y. Shifrin, E. K. Grishaw, and A. R. Padhani, "Virtual reality imaging of human anatomy," in *SCAR 94: Computer Applic. to Assist Radiol.*, pp. 703–708, J.M. Boehme, A.H. Rowberg, N.T. Wolfman, eds., 1994.

[3] W. E. Higgins and K. Ramaswamy, "Toward dynamic visualization for endoscopy simulation," *Proc.* 16^{th} *IEEE Engin. in Med. and Biol. Soc.*, vol. 1, Baltimore, MD, pp. 700–701, Nov. 1994.

[4] W. E. Lorensen, F. A. Jolesz, and R. Kikinis, "The exploration of cross-sectional data with a virtual endoscope," *Interactive Technology and the New Health Paradigm*, pp. 221–230, Jan. 1995.

[5] L. Hong, A. Kaufman, Y. Wei, A. Viswambharan, M. Wax, and Z. Liang, "3D virtual colonoscopy," in *Proc. Biomed. Visual. '95,*, M. Loew and N. Gershon, eds., Atlanta, GA, pp. 26–32 and 83, Oct. 1995.

[6] K. Mori, J. Hasegawa, J. Toriwaki, H. Anno, and K. Katada, "Automatic extraction and visualization of bronchus from 3D CT images of lung," *Lecture Notes in Computer Science*, vol. 905, pp. 542–548, N. Ayache (ed.), 1995.

[7] A. K. Hara, C. D. Johnson, J. E. Reed, D. A. Ahlquist, H. Nelson, R. L. Ehman, C. M. McCollough, and D. M. Ilstrup, "Detection of colorectal polyps by computed tomographic colography: feasibility of a novel technique," *Gastroenter.*, vol. 110, no. 1, pp. 284–290, Jan. 1996.

[8] K. H. Höhne, B. Pflesser, A. Pommert, M. Riemer, T. Schiemann, R. Schubert, and U. Tiede, "A 'virtual body' model for surgical education and rehearsal," *Computer*, vol. 29, no. 1, pp. 25–31, Jan. 1996.

[9] K. Ramaswamy and W. E. Higgins, "Endoscopic exploration and measurement in 3D radiological images," in *SPIE Medical Imaging 1996: Image Processing*, vol. 2710, Newport Beach,CA, pp. 511–523, 10-15 Feb. 1996.

[10] P. Haigron, G. Le Berre, and J. L. Coatrieux, "3D navigation in medicine," *IEEE Engin. Med. Biol. Mag.*, vol. 15, no. 2, pp. 70–78, Mar.-Apr. 1996.

[11] D. J. Vining, R. J. Zagoria, K. Liu, and D. Stelts, "CT cystoscopy: an innovation in bladder imaging," *Am. J. Roentgen.*, vol. 166, no. 2, pp. 409–410, Feb. 1996.

[12] D. J. Vining, K. Liu, R. H. Choplin, and E. F. Haponik, "Virtual bronchoscopy: relationships of virtual reality endobronchial simulations to actual bronchoscopic findings," *Chest*, vol. 109, no. 2, pp. 549–553, Feb. 1996.

[13] G. D. Rubin, C. F. Beaulieu, V. Argiro, H. Ringl, A. M. Norbash, J. F. Feller, M. D. Dake, R. B. Jeffrey, and S. Napel, "Perspective volume rendering of CT and MR images: applications for endoscopic imaging," *Radiology*, vol. 199, no. 2, pp. 321–330, May 1996.

[14] R. M. Summers, D. H. Feng, S. M. Holland, M. C. Sneller, and J. H. Shelhamer, "Virtual bronchoscopy: Segmentation method for real-time display," *Radiology*, vol. 200, no. 3, pp. 857–862, Sept. 1996.

[15] R. M. Summers, "Navigational aids for real-time virtual bronchoscopy," *Am. J. Roentgenology*, vol. 168, no. 5, pp. 1165–1170, May 1997.

[16] C. L. Kay and H. A. Evangelou, "A review of the technical and clinical aspects of virtual endoscopy (review)," *Endoscopy*, vol. 28, no. 9, pp. 768–775, Nov. 1996.

[17] W. E. Higgins, K. Ramaswamy, R. Swift, G. McLennan, and E. A. Hoffman, "Virtual bronchoscopy for 3D pulmonary image assessment: State of the art and future needs," *Radiographics*, vol. 18, no. 3, pp. 761–778, May-June 1998.

[18] D. J. Hawkes, D. L. G. Hill, C. Studholme, P. J. Edwards, and W. L. Wong, "Three-dimensional multimodality imaging in image-guided interventions," *Seminars in Intervent. Radiol.*, vol. 12, no. 1, pp. 63–74, March 1995.

[19] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *Computer Graphics*, vol. 22, no. 4, pp. 65–74, Aug. 1988.

[20] A. Kaufman, *Volume Visualization.* Los Alamitos, CA: IEEE Computer Society Press, 1991.

[21] A. Kaufman, "Volume graphics," *Computer*, vol. 26, no. 7, pp. 51–64, July 1993.

[22] R. Shahidi, B. Lorensen, R. Kikinis, J. Flynn, A. Kaufman, and S. Napel, "Surface rendering versus volume rendering in medical imaging: techniques and applications," *Symp. Volume Visualization 1996*, pp. 439–440, 1996.

[23] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proc. 1994 Symp. Volume Visualization*, Washington, D.C., pp. 91–98, Oct. 1994.

[24] B. Corrie and P. Mackerras, "Parallel volume rendering and data coherence," in *Proc. 1994 Symp. Volume Visualization*, pp. 23–26, Oct. 1994.

[25] T. A. Foley, D. A. Lane, and G. M. Nielson, "Towards animated ray-traced volume visualization," *J. Visual. Comput. Anim.*, vol. 1, pp. 2–8, 1990.

[26] T. Malzbender, "Fourier volume rendering," *ACM Trans. on Graphics*, vol. 12, no. 3, pp. 233–250, July 1993.

[27] T. K. Wu and M. L. Brady, "Parallel approximate computation of projections for animated volume-rendered displays," in *Proc. 1993 Parallel Rend. Symp.*, pp. 61–66, Oct. 1993.

[28] H. W. Shen and C. R. Johnson, "Differential volume rendering: A fast volume visualization technique for flow animation," in *Proc. IEEE Visualization 94*, pp. 188–195, 1994.

[29] D. Laur and P. Hanrahan, "Hierarchical splatting: A progressive refinement algorithm for volume rendering," *Computer Graphics*, vol. 25, no. 4, pp. 285–288, July 1991.

[30] S. Badt, Jr., "Two algorithms for taking advantage of temporal coherence in ray tracing," *The Visual Computer*, pp. 123–132, 1988.

[31] J. Chapman, T. W. Calvert, and J. Dill, "Exploiting temporal coherence in ray tracing," in *Proc. Graphics Interface 1990*, pp. 196–204, 1990.

[32] B. Gudmundsson and M. Randen, "Incremental generation of projections of CT volumes," in *Proc. Visual. in Biomed. Computing*, Atlanta, GA, pp. 27–34, May 1990.

[33] R. Yagel and Z. Shi, "Accelerating volume animation by space-leaping," in *Proc. IEEE Visualization 93*, pp. 226–231, Oct. 1993.

[34] J. E. Reed and C. D. Johnson, "Automatic segmentation, tissue characterization, and rapid diagnosis enhancements to the computed tomographic colonography analysis workstation," *SPIE Medical Imaging '97: Physiol. and Func. from Multidim. Images*, vol. E. A. Hoffman (ed.), v. 3033, pp. 433–440, 1997.

[35] R. A. Robb and C. Barillot, "Interactive display and analysis of 3-D medical images," *IEEE Trans. Medical Imaging*, vol. 8, no. 3, pp. 217–226, Sept. 1989.

[36] U. Tiede, K. H. Hoehne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke, "Investigation of medical 3D rendering algorithms," *IEEE Computer Graph. and Applic.*, vol. 9, no. 2, pp. 41–53, Mar. 1990.

[37] D. R. Ney, E. K. Fishman, D. Magid, and R. A. Drebin, "Volumetric rendering of computed tomography data: Principles and techniques," *IEEE Trans. Computer Graph. & Applic.*, vol. 9, no. 2, pp. 24–32, 1990.

[38] J. Wilhelms and A. V. Gelder, "A coherent projection approach for direct volume rendering," *Computer Graphics*, vol. 25, no. 4, pp. 275–283, July 1991.

[39] R. D. Swift, K. Ramaswamy, and W. E. Higgins, "Adaptive axes-generation algorithm for 3D tubular structures," *1997 IEEE Int. Conf. Image Processing*, vol. II, pp. 136–139, 1997.

[40] R. D. Swift, W. E. Higgins, E. A. Hoffman, G. McLennan, and J. M. Reinhardt, "Automatic axis generation for 3D virtual-bronchoscopic image assessment," *SPIE Medical Imaging 1998: Physiology and Function from Multidimensional Images*, vol. 3337, E.A. Hoffman (ed.), 1998.

[41] H. Haneishi, Y. Yagihashi, and Y. Miyake, "A new method for distortion correction of electronic endoscope images," *IEEE Trans. Medical Imaging*, vol. 14, no. 3, pp. 548–555, Sept. 1995.

[42] K. Ramaswamy and W. E. Higgins, "Coherence-based dynamic navigation for virtual endoscopy," *SPIE Medical Imaging 1998: Image Display*, vol. vol., 3335, Y. Kim and S. K. Mun (eds.), pp. 410–426, 1998.

Table 1: Forward translation using a perspective-projection geometry

Starting site	Brute force		Coherence	
	T_1	T_2	T_1	T_2
$z = 40$	0.197	0.211	0.121	0.134
$z = 50$	0.185	0.198	0.052	0.066
$z = 60$	0.127	0.141	0.024	0.036
$z = 70$	0.096	0.109	0.024	0.035

Table 2: Forward translation using a parallel-projection geometry

Starting site	Brute force		Coherence	
	T_1	T_2	T_1	T_2
$z = 40$	0.203	0.215	0.011	0.022
$z = 50$	0.168	0.181	0.011	0.022
$z = 60$	0.138	0.150	0.011	0.022
$z = 70$	0.075	0.088	0.011	0.022

Table 3: Viewing-site rotations using a perspective-projection geometry

Viewing site	Brute force		Coherence	
	T_1	T_2	T_1	T_2
$z = 40$	0.119	0.126	0.105	0.112
$z = 50$	0.108	0.115	0.055	0.062
$z = 60$	0.091	0.098	0.027	0.034
$z = 70$	0.061	0.068	0.018	0.024

Table 4: Viewing-site rotations using a parallel-projection geometry

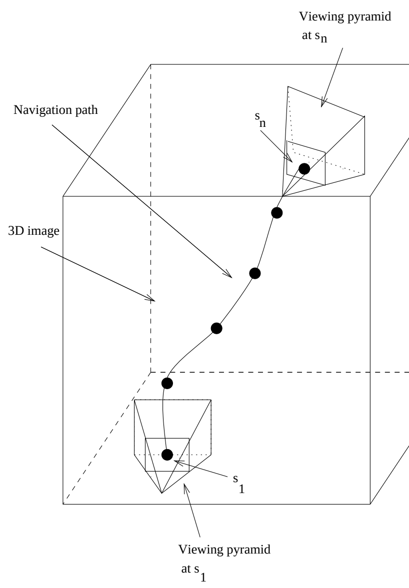
Viewing site	Brute force		Coherence	
	T_1	T_2	T_1	T_2
$z = 40$	0.108	0.115	0.022	0.027
$z = 50$	0.111	0.118	0.012	0.019
$z = 60$	0.091	0.097	0.012	0.019
$z = 70$	0.066	0.073	0.012	0.019

Table 5: Errors in computed depths when performing forward and backward translations with a perspective viewing geometry.

Translation index	Forward		Backward	
	e_{nn}	e_t	e_{nn}	e_t
10	0.39	0.65	0.04	0.53
20	0.14	0.36	0.24	0.57
30	0.03	0.27	0.26	0.52
40	0.01	0.29	0.36	0.64
50	0.02	0.40	0.27	0.58
60	0.09	0.41	0.24	0.41
70	0.14	0.38	0.25	0.39
80	0.11	0.53	0.12	0.29
90	0.28	0.60	0.08	0.29
100	0.50	0.62	0.10	0.27

Table 6: Errors in computed depths when performing viewing-site rotations using parallel and perspective viewing geometries.

rotation index	Parallel		Perspective	
	e_{nn}	e_t	e_{nn}	e_t
10	0.24	0.54	0.24	0.42
20	0.28	0.68	0.21	0.50
30	0.26	0.68	0.41	0.54
40	0.33	0.71	0.54	0.78
50	0.42	0.75	0.58	0.71
60	0.25	0.51	0.57	0.62
70	0.23	0.60	0.62	0.65
80	0.31	0.64	0.64	0.62
90	0.31	0.65	0.30	0.38
100	0.27	0.62	0.10	0.21



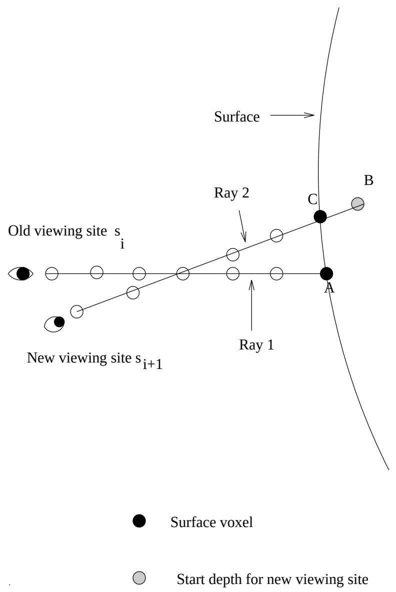


Figure 1: (left) Navigation geometry. The large cube represents the 3D digital image environment. The curved line depicts a possible 3D navigation path $\mathbf{s}_1, \ldots, \mathbf{s}_n$ traversed through the image. Each small pyramid represent the viewing pyramid (visible data) for each viewing site. Note that the viewing pyramid reduces to a parallelepiped for a parallel-projection viewing geometry.

Figure 2: (right) Illustration of computational savings obtained using the coherence method, shown for a single ray at viewing-screen location (j, k). Using brute-force ray-casting at $\mathbf{s}_i$, a distance of 6 ray samples has to be traveled before the surface is struck. If, instead, the coherence algorithm is used at $\mathbf{s}_{i+1}$, starting at depth $\rho_i^{strike}[j,\ k]$, a distance of one ray sample has to be traveled before the surface is struck.

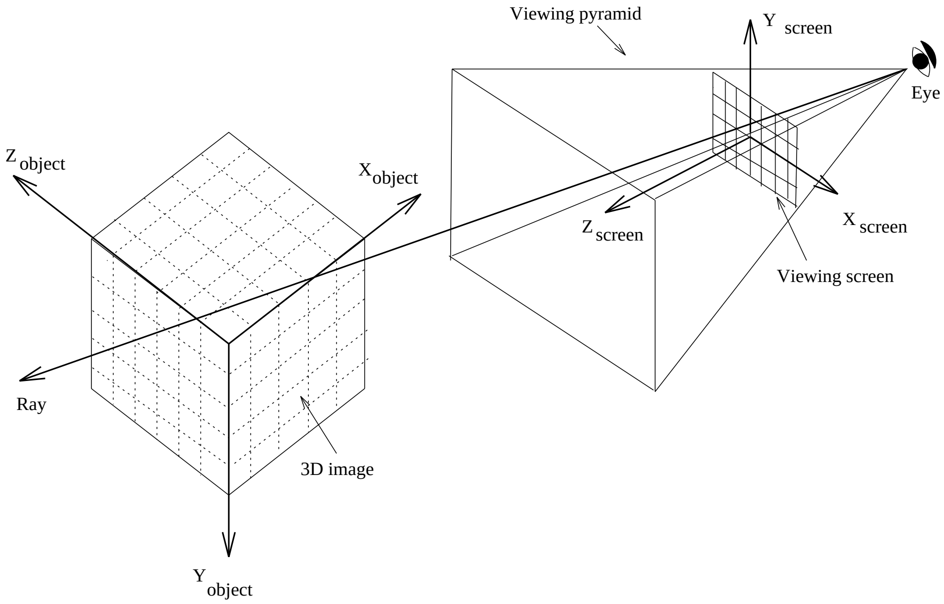


Figure 3: Ray casting model used in our method.

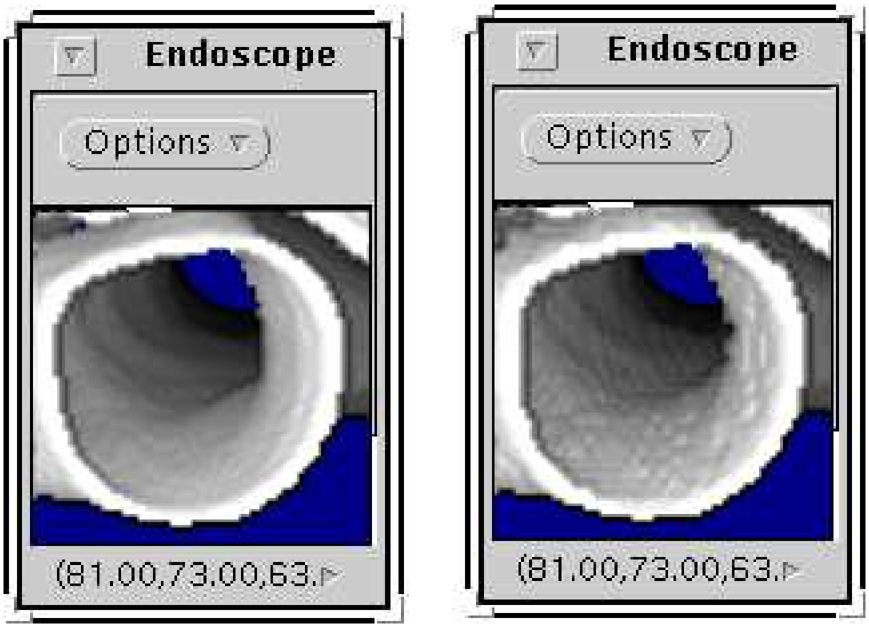
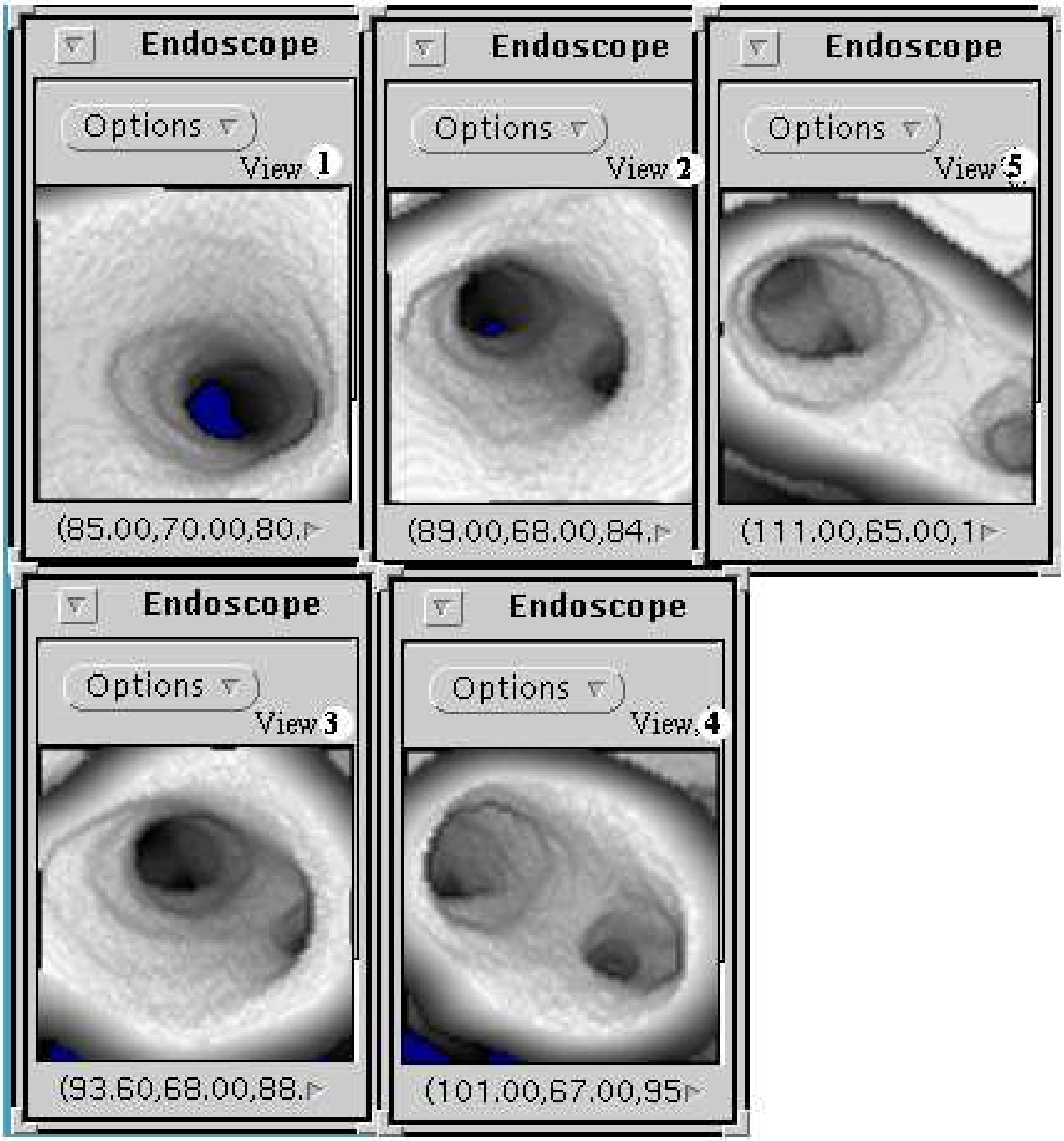


Figure 4: Comparison of endoscopic views computed using brute-force and coherence methods. Part (a) shows the view computed using a brute-force rendering method while part (b) shows the view computed using the coherence method in Algorithm 3. The views were computed using depth-gradient shading, and correspond to the results shown for trilinear-interpolation sampling and parallel-projection geometry in Row 3 of Table 6.



Part of Figure 5.

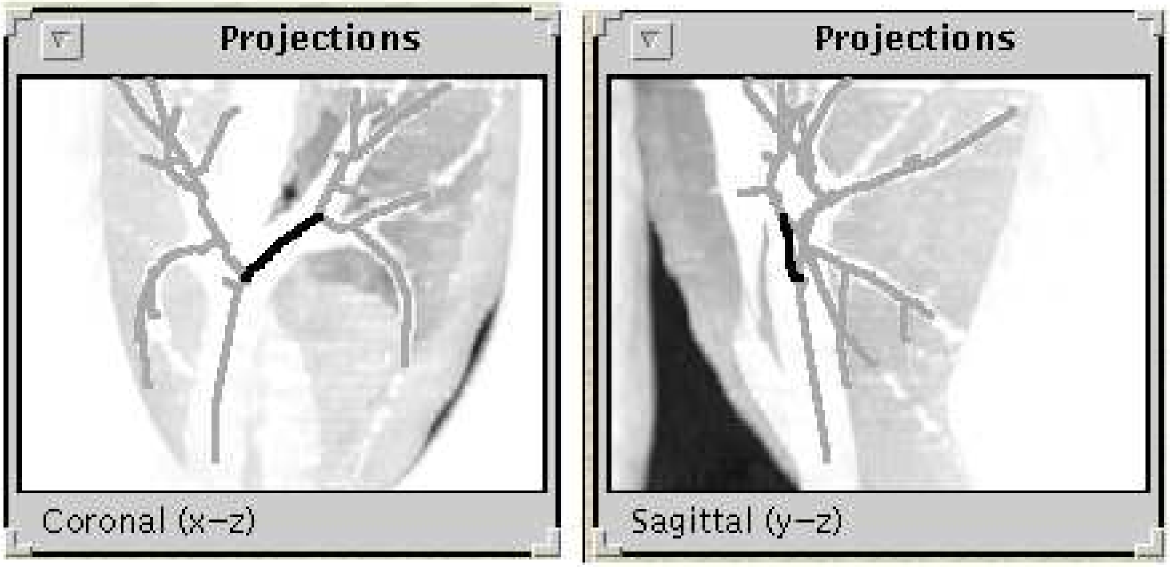
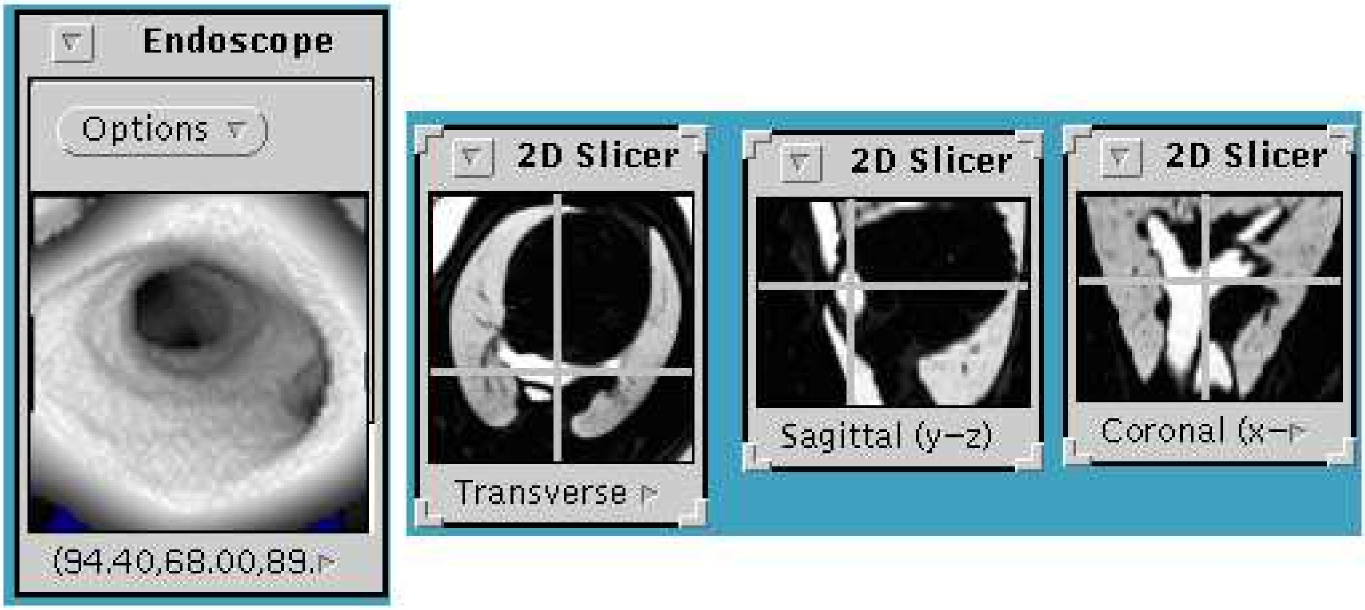


Figure 5: Interactive viewing of a path trajectory created using automatic processing. The bright branching structures on the Coronal and Sagittal *Projections* views are the pulmonary airways (3D image gray scale was inverted for clarity). Gray lines indicate the path trajectories through the pulmonary airways, computed using automatic processing. The selected branch is darkened on *Projections* displays. The *Endoscope* shows selected volume-rendered views from the movie computed along the path using the coherence method. *Endoscope* parameters: double-sized 64 $\times$ 64 viewing screen, $r_{max} = 64$, min. rendering threshold = 40, max. rendering threshold = 215, perspective viewing geometry ($z_e^s = 30$, FOV = 93^o), depth-gradient-plus-gray shading, and a zoom of 30 %.



Part of Figure 6.

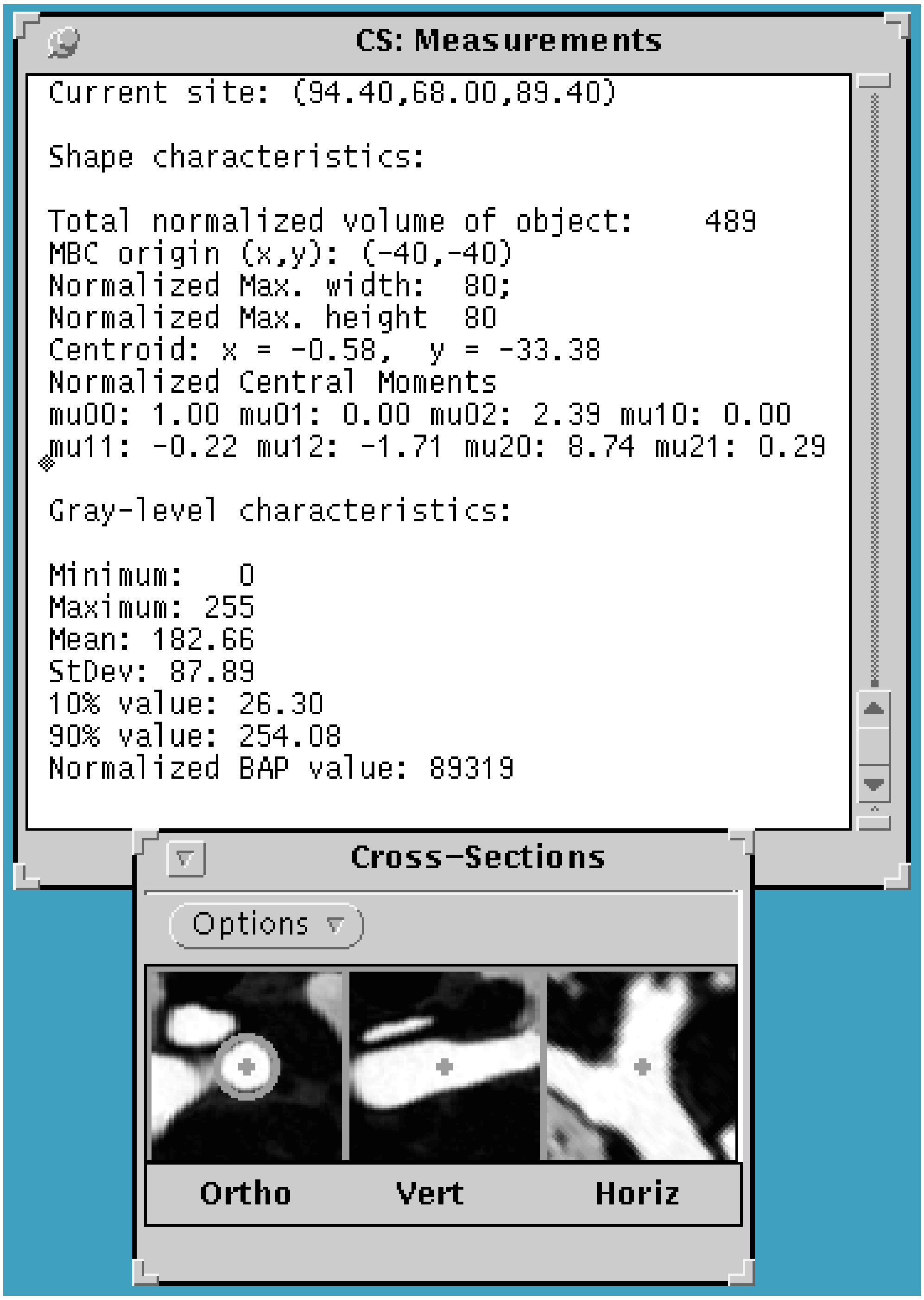
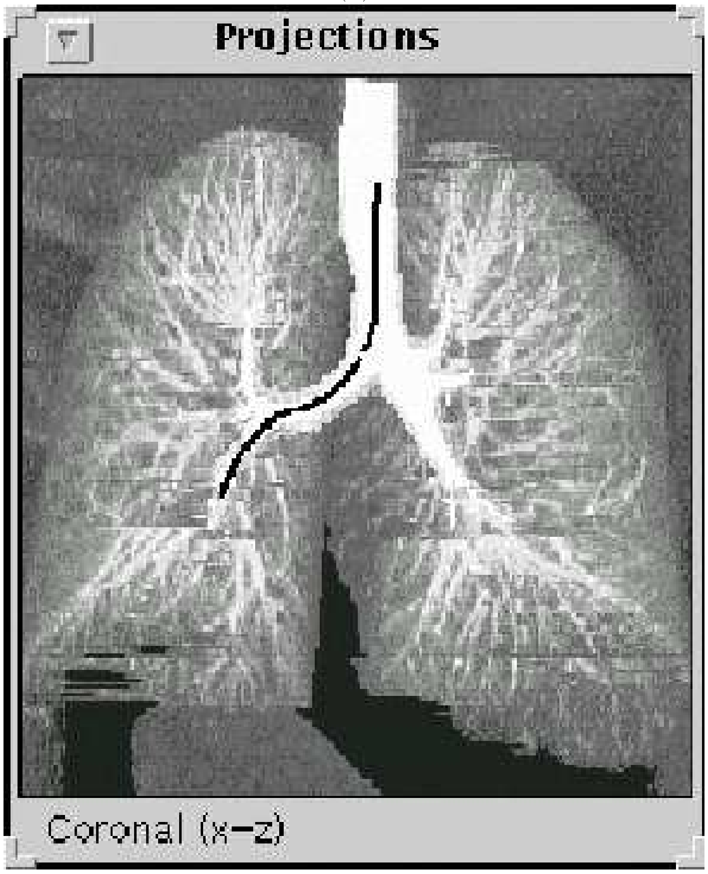


Figure 6: *Endoscope, 2D Slicer, Cross-Sections, Measurements* tool. The tools are computed at a viewing site in the right lung's middle lobe. The *Endoscope* shows a volume-rendered view computed using the coherence method. (Same *Endoscope* parameters as in the previous figure.) The *2D Slicer* shows three orthographic planes of data from the 3D image passing through the current viewing site. The intersection of the cross-hairs on the *2D Slicer* displays gives the current viewing site. The *Cross-Sections* tool shows the viewing site as a "+" symbol. The displays show three mutually perpendicular planes of data, relative to the current viewing direction, from the 3D image. Notice how the *Vert* and *Horiz* displays track the airway. The region inside the circular disc on the *Ortho* panel is used for making various gray-scale and shape measurements, as shown in the *Measurement* tool.

(a)



(b)

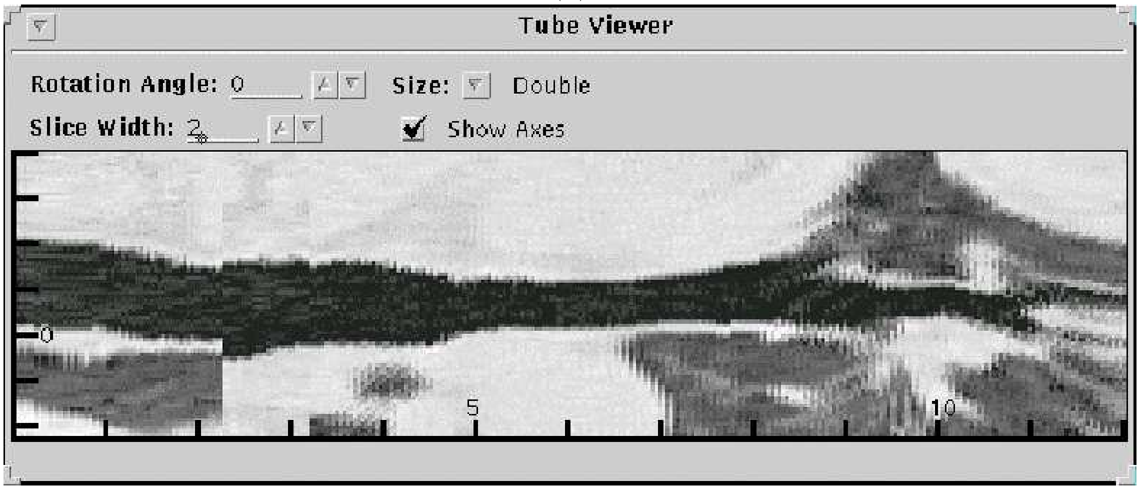
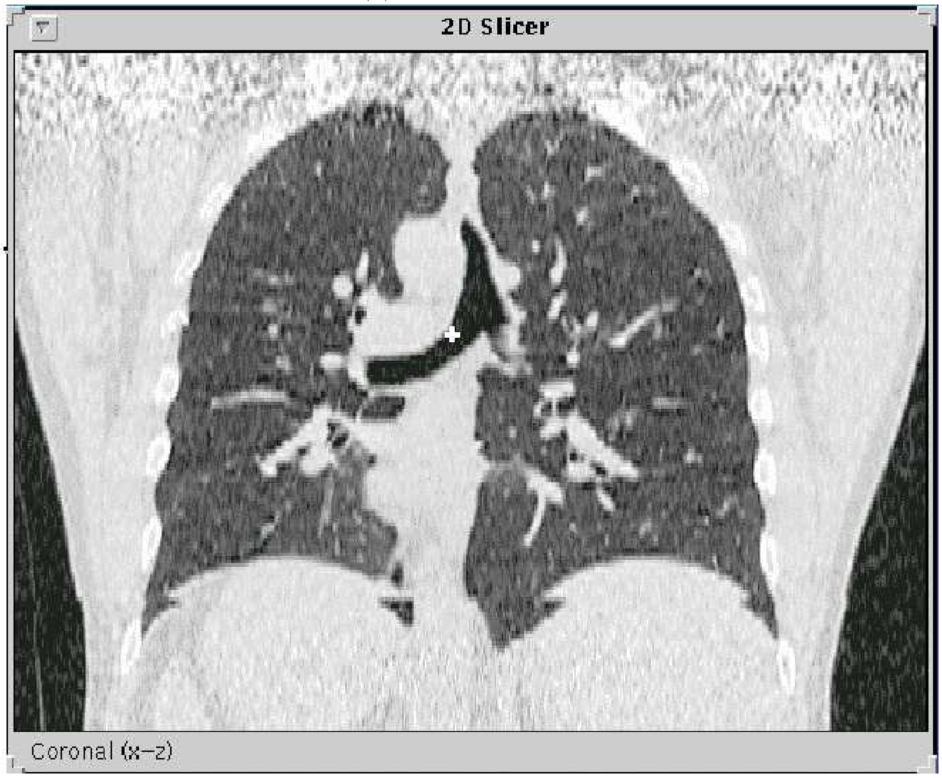


Figure 7: Virtual-endoscopic system applied to a healthy human volunteer. (a) [top]: Coronal maximum-intensity projection of 3D CT image; the black line shows a path trajectory through the airways and the white site highlighted on the path is examined in parts (c-d). (b) [bottom]: mathematically straightened ("tube") view of gray-scale data along path trajectory; dark area is air. *Next Page:* (c) [top]: global x-z coronal slice cut through 3D image; the white '+' indicates one site along the highlighted trajectory. (d) [bottom left]: Endoscope shows interior rendered view at highlighted '+' sight of 2D Slicer view in (c). (e) [bottom right]: Helm tool for manual navigation; as shown, many directional movements are possible.

(c)

Coronal...



(d)

(e)

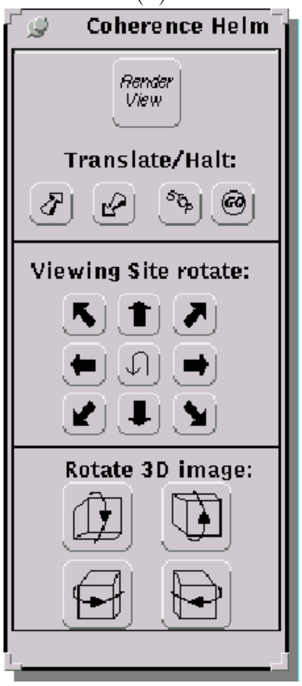


Figure 7 concluded.

SUMMARY

A high-resolution, 3D, radiologic image provides a virtual copy of the anatomy. Complex anatomical information can be obtained from the image by navigating through it in a manner similar to an endoscopic examination. Real-time computation of these "virtual" endoscopic views, however, is needed to permit interactive navigation. We present a fast volume-rendering method for computing such views and illustrate the method's use in a complete system for virtual endoscopy.

Our method, motivated by the temporal-coherence concept, performs dynamic volume rendering at interactive frame rates. The method consists of a series of six straightforward algorithms. Algorithm 1 gives a top-level synopsis of the method. Algorithms 2-6 are called within the body of Algorithm 1. To begin navigation at viewing site $\mathbf{s}_1$ (Algorithm 2), a brute-force calculation must first be done; initial depth information is recorded and the initial view V_1 is computed. For subsequent viewing sites $\mathbf{s}_i$, $i = 2, 3, \ldots$, along a navigation path, the depth information is updated and new views are computed quickly. Algorithms 3 and 4 are general coherence algorithms for rendering views corresponding to any incremental forward or backward movement along a slowly changing navigation path. Algorithms 5 and 6 discuss improved algorithms for rendering views corresponding to pure translational movements made using a parallel-projection viewing geometry.

Timing results show that the coherence method greatly outperforms brute-force ray casting in all cases, often on the order of a factor of ten. The greatest speed gain over brute-force rendering is achieved when performing pure translations using a parallel-projection geometry. For the parallel geometry, the coherence method's rendering time rivals the time needed to complete the display of the image. The coherence method gives results ranging from 8 frames/sec to 52 frames/sec. With current systems running at 300 Mhz and higher, these rates can be much higher and clearly in the realm of comfortable real-time interaction. Other results demonstrate the method's accuracy.

The method constitutes part of a computer-based virtual-endoscopic system we have devised. The system has many tools that can be used during interactive navigation. Because the virtual endoscopic views can be quickly with our method, navigation with the complete system runs interactively. The system is illustrated using high-resolution 3D pulmonary CT images of a dog and a human volunteer.