VIRTUAL BRONCHOSCOPY

Pheng-Ann Heng, Ping-Fu Fung,
Kwong-Sak Leung, and Han-Qiu Sun
The Chinese University of Hong Kong

Tien-Tsin Wong
Hong Kong University of Science & Technology

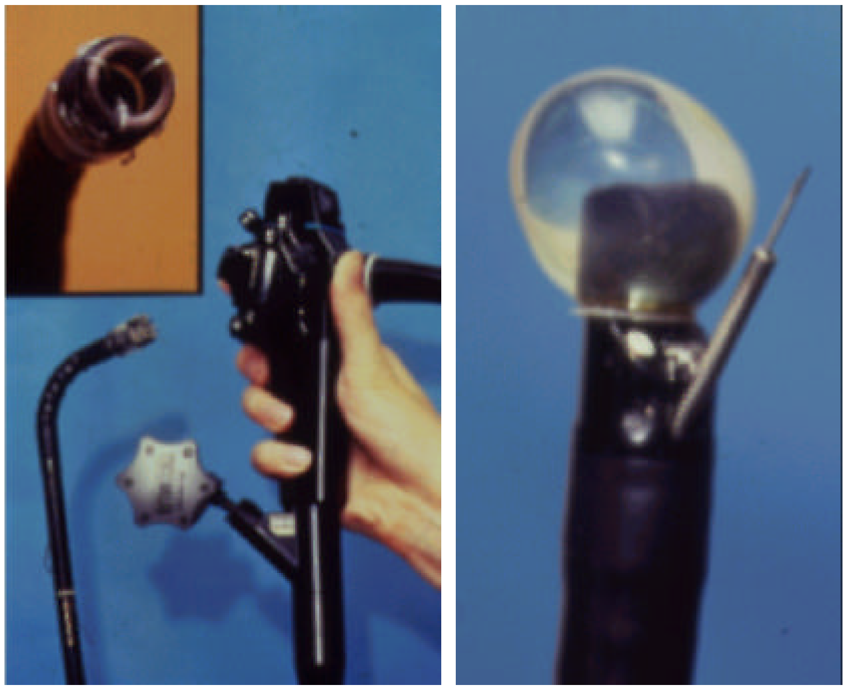
ABSTRACT

Patients potentially suffer and are exposed to danger during invasive bronchoscopic diagnosis and surgery. In order to reduce this hazardous risk, we have developed an interactive virtual environment for the simulation of bronchoscopy (in short, called "virtual bronchoscopy"). Because of this state-of-the-art application, medical doctors can now obtain pre-operative information and perform pilot examinations in a virtual environment without any invasive or needless surgery. This 3D lung volume data of the patient is first acquired from CT and/or MRI scanning, without any pain being inflicted upon the patient. Then a vessel-tracking process is used to extract the patient's bronchial tree from the data. It is important to note that while manual tracking is tedious and labor-intensive, fully automatic tracking may not be as reliable in such a critical medical application. Thus a semi-automatic tracking technique called the *Intelligent Path Tracker*, which provides automation and sufficient user control during the tracking process, is most useful. This methodology is applied to a virtual bronchoscopy session, where doctors can use a 3D pen input device to navigate and visualize the bronchial tree of patients in a natural and interactive manner. To support an interactive frame rate, we also propose a new volume rendering acceleration technique, named *IsoRegion Leaping*. Through this technique visualization is further accelerated using a distributed rendering process based upon a TCP/IP network of low-cost PCs. Combining these approaches enables a 256x256x256 volumetric data representation of a human lung to be navigated and visualized at a frame rate of over 10 Hz in our virtual bronchoscopy system.

1. Introduction

The recent development of computer visualization technology meets the ever-growing need of more advanced tools for medical diagnoses, surgical operations, and training. We have developed a system for computer simulation of bronchoscopy, and this system provides interactive navigation capability, allowing the user to rapidly fly through the airway branches. One of the key focal points in designing and implementing our system is the user interface, mainly because it's goal is to mimic the control of a real bronchoscope and provide sufficient information to the user interactively.

Bronchoscopy is basically defined as a medical diagnosis for investigating bronchial tube and tracheobronchial tree diseases. In practice, doctors/surgeons insert a bronchoscope, which is a tube-like instrument, into the airway of the patient through the throat. At the tip of the bronchoscope is a tiny camera and an illuminating element where optical signal transmissions are processed. Typically these signal connections are made with fibre optics that are especially designed to be slim. At the opposite end of the bronchoscope a surgeon holds the control knobs. Through the video system, the surgeon can view and/or record the image captured by the bronchoscope. Using this procedure the surgeon can navigate the device in realtime by moving the scope back and forth, as well as by rotating and turning it. Figure 1 shows an overall view of a real bronchoscope along with a close-up view of the tip.



(a) Clinical bronchoscope *(b) Scope tip*

Figure 1 Pictures of a clinical bronchoscope.

When there is a severe constriction in the air passsage, clinical bronchoscopy carries a certain risk of worsening the situation. The result may be hypoxia due to coughing or anxiety caused by the procedure itself. As one can see, a totally non-invasive technique providing quick and accurate pre-operative information for the planning of interventional bronchoscopic procedures would be very useful.

One key advantage of our system is that it is equipped to provide this type of information. Another advantage is that trainees can learn and enhance their operative skill level through simulation techniques before working on human beings.

This system consists of two main sub-systems which include interactive visualization and navigation, as well as vessel tracking. The interactive visualization and nagivation sub-system allows the doctor to navigate freely within the volume. However, moving freely without any directional navigation in the volume may not be helpful. For instance, he/she may want more detailed information, such as the appearance of an actual bronchoscope moving within an actual bronchial tube. For this reason more constrained navigation is sometimes necessary. To accomplish this we need to know the exact location of the bronchial. In order to extract the information about the bronchial tree, we have developed an intelligent path tracking sub-system which allows the doctor to track bronchial tubes accurately and conveniently. Here a semi-automatic approach has been implemented within the sub-system which allows the bronchial tube to be tracked. This technique is more reliable than fully automatic tracking, as it allows the user to exercise some control. Moreover, it is more convenient than tedious manual tracking. The doctor only needs to specify the start and end points of the tip, and the system then automatically finds the path connecting them.

The following sequence of our discussion is as follows: Section 2 discusses some of the previous systems, and the user interface of the system is briefly introduced in Section 3. Sections 4 and 5 discuss two acceleration approaches used in the interactive visualization and navigation sub-system to achieve interactive frame rate. Section 6 describes the intelligent path tracking sub-system. Finally, some conclusions and future directions are described in Section 7.

2. Previous Work and Our Goals

Several medical visualization systems have been proposed for visualizing various parts of the human body. They include brain [19], heart [18][20], colon [6], lung, and the bronchial tree [16]. Ney *et al.* [14] developed a system for visualizing the branching structures (external view) of the bronchial tree. However, this system does not support interactive navigation. Researchers in CieMed of National University of Singapore proposed the Virtual Workbench [16] and Cardiac Visualization Systems [20][12], which can generate images at a rate of about 10 Hz. However, they are strongly dependent on the 3D texture mapping hardware that are usually expensive. Moreover, the data set size is also limited. A simulation for endoscopic procedures is described by Geiger and Kikinis [5], and this system employs geometric model and surface rendering techniques. With surface collision detection and physical models, the system calculates the trajectory of a flythrough. However, it does not support interactive navigation, and the frame rate is only about 1 Hz. Our aim is to develop a computer simulation system for bronchoscopy which supports volume visualization and navigation at an interactive speed with general-purpose and low cost hardware. The following properties are required:

- **High performance** - interactive speed
- **Cost effective** - affordable to most medical institutions
- **User friendly** - easy to use, not complicated
- **Portable** - runs readily on most platforms

3. User Interface Design

3.1 IMAGE VIEW

In a real bronchoscopy operation the doctor can monitor the patient through the camera tip. There are also other aids, such as an a X-ray caster revealing the position of the bronchoscope tip. Likewise, there are other image views in our system to help the user to acquire related and useful information during a simulated bronchoscopy.

Airway view (Direct volume rendering)

This is the most important view because it displays what the bronchoscope tip captures. In addition, volume rendering is used to simulate the view. However, it costs tremendous computation time. Improved rendering algorithm and parallel computation are employed in an effort to increase the speed of this view. Those techniques will be discussed in Sections 4 and 5.

Cut-plane view (Single sample ray casting

This view presents an cross-sectional slice in the volume. The view-plane is defined by the position and orientation of the camera tip, and this is considered to be the cutting plane perpendicular to the camera tip. Due to its small computational time requirement, this approach provides the user with an extremely fast approximation to the camera tip view, which gives the user a rough idea as to the appearance of the actual camera tip. This is especially advantageous when not enough computational power can be assigned to help generate the camera tip view. This approach provides a fast view finder that enables the user to locate the view point when getting lost.

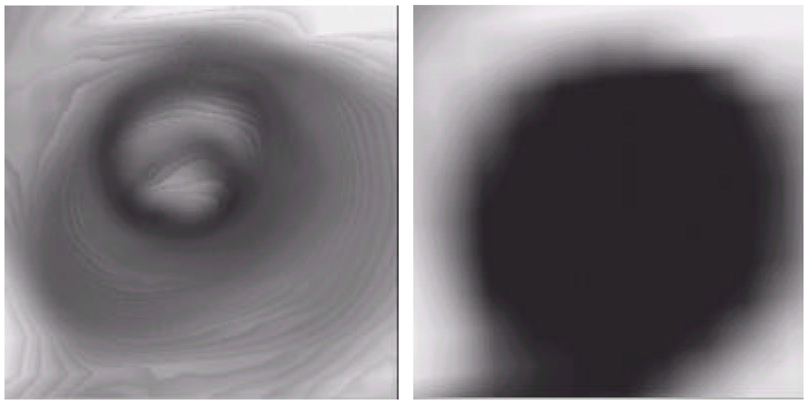
Tri-plane view (Z-sorted surface ray casting)

By cutting the volume along three principal orthogonal directions, we get three cutting plane images. Combining the three images according to their occlusion order results in a tri-plane view. This view is very useful in locating the position of the camera tip when the intersection point of the three planes is located at the camera tip. The user may also move the planes freely to visualize the internal volume and locate some branches.

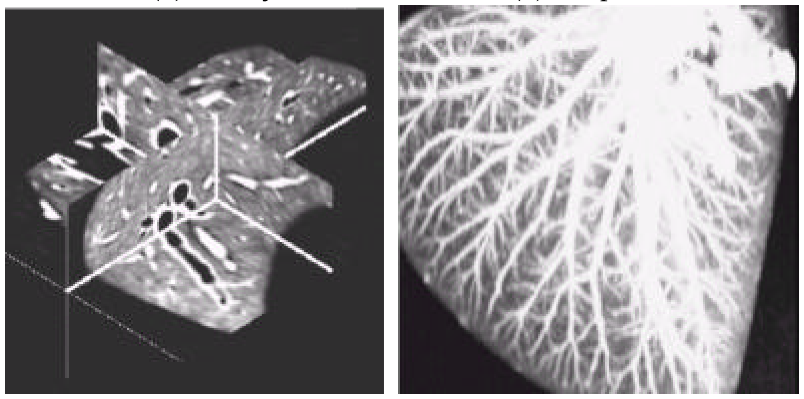
X-ray views (Maximal intensity projection)

This is a set of views presenting X-ray (maximal intensity projection) views of the volume along the three principal orthogonal directions. It is a fast tool for locating the camera tip position since the tip can be highlighted. It simulates the X-ray projection view in clinical bronchoscopy operation.

Figure 2 illustrates the four views. Figure 3 illustrates a screen shot showing a virtual bronchoscopy session of a lung. The data set dimensions are 256 by 256 by 185. Each image window has a size of 256 by 256.



(a) airway (b) cut-plane



(c) tri-plane (d) X-ray

Figure 2 Four kinds of views.

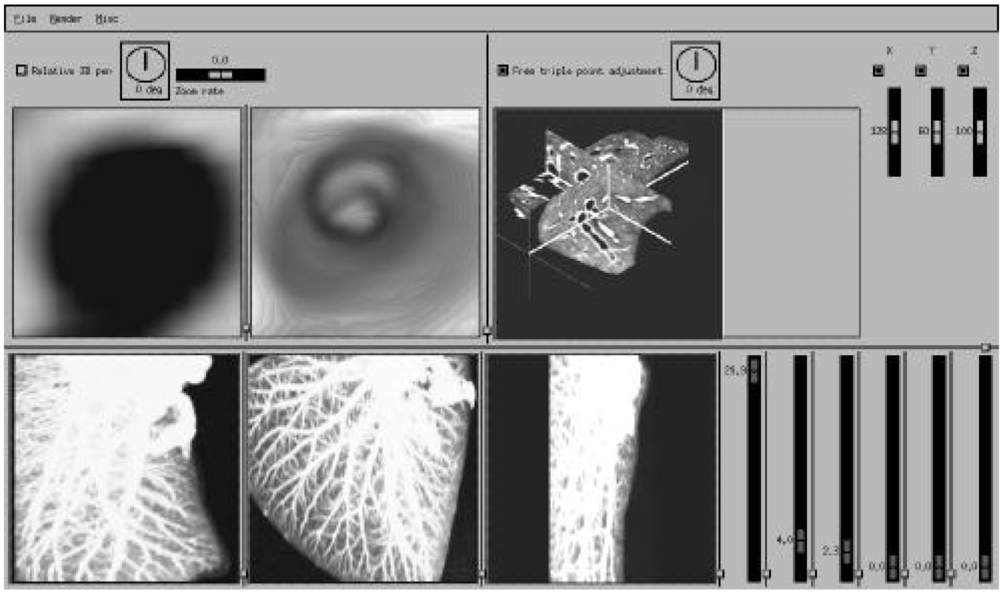


Figure 3: A screen shot showing a virtual bronchoscopy session..

3.2 NAVIGATION WITH 3D PEN

A special input device, called a 3D pen, is employed in our system. Since our system is a 3D visualization application, it is natural and convenient to adopt such an input device. During virtual operation our system requires the user to specify the 3D camera position and orientation.

We use a Polhemus 3 space tracker system [15] which is a 3D electromagnetic tracking device consisting of a pair of transmitters and receivers. The system is able to detect the 3D position and orientation of the receiver relative to the transmitter. The data consists of numerical values in millimeters for the position corresponding to the X, Y and Z axes, plus angles in degree for the roll, pitch and yaw orientations. The receiver is shaped like a pen, and looks like the tip of a bronchoscope. This provides a more intuitive way of controlling the virtual bronchoscope. (See Figure 4).

Since the 3D pen input device can be moved freely in space, it allows the user to control arbitrarily with 6 degrees of freedom. Besides moving freely in the volume, which helps the doctor confine the virtual bronchoscope to the muscular tube-like branching structure during simulation, this can also prevent the doctor from getting lost within the volume. The system provides a directional path movement control that restricts movement along a bronchial tube.

In addition, if the input device provides relative movement control, it will definitely help keep the navigation smooth. With absolute control the user has a greater freedom of moving around to search for tissue or the option of where to place the virtual camera. However, as mentioned previously, this could lead to loss of orientation of the bronchial scope.

This system is advantageous because both features have been incorporated. The user can easily switch between these two modes of operation by use of a single button click.

Figure 4: A user interactively controls our system.

4. IsoRegion Leaping Acceleration

Due to the amount of volume data, volume rendering in real-time is not feasible without some acceleration technique. In this section we describe an algorithm, known as *IsoRegion leaping*, which aids in acceleration of the rendering process**.** The next section describes how this process can be accelerated using distributed volume rendering. By applying these two approaches, we can achieve an interactive frame rate.

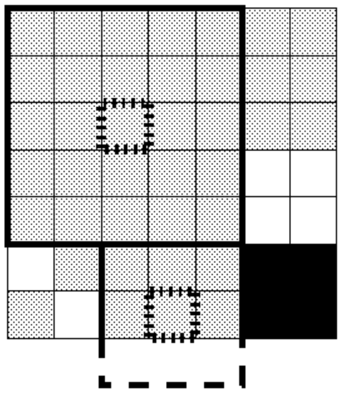
4.1 SPACE LEAPING

Space leaping is a well-known acceleration technique for ray-casting type volume rendering by skipping empty spaces [22]. There are several variants such as proximity-clouds [1], ray acceleration by distance coding [23], and macro-regions of empty space [21]. All of these space leaping techniques accelerate the ray casting process by avoiding unnecessary, time-consuming, empty space traversal. To apply the algorithms one has to first classify the empty space (background voxels) by means of thresholding, segmentation, and/or statistical methods. However, the classification process often introduces inaccurate results and requires human intervention.

Unlike existing algorithms that only skip empty space, voxel volumes with the same voxel values are also eligible to be leaped under the concept of voxel block leaping, leading to further speedup. Such voxel volumes are called *IsoRegions*. With IsoRegions, we not only identify the empty space, but also volumes with the same value so that we can leap over them. During ray casting, samples are taken on equal intervals along the ray path. Because the ray samples within an IsoRegion take the same voxel value, pre-computation of the ray segment colors and transparencies is possible, and this technique is the main source of effiiency and improvement in the model.

An IsoRegion is an odd dimension cubic volume of voxels with the same voxel value. Each IsoRegion consists of $(2\boldsymbol{d}+1)^3$ voxels for some non-negative integer ***d***. Hence the smallest IsoRegion contains only one voxel, which is a specialized case with ***d***=0. This definition ensures the voxel located at the center of the IsoRegion is surrounded by 26 connected neighborhoods with the same voxel value, which has a range of ***d*** steps. Such a 26 connected neighborhood distance is also known as "chess board distance" [21]. It indicates what the least voxel grid distance would be in order to go from one voxel to the nearest heterogeneous voxel. From the center of an IsoRegion we can leap over ***d*** voxel grid distance in all directions to get to the boundary of the IsoRegion directly.

To store the IsoRegion information, only the dimension parameter ***d*** of the IsoRegion is recorded at the center of that IsoRegion. Figure 5 shows an example for the 2D case. Note that the IsoRegions are constructed for all voxel values at once.



4	3	2	2	2	2	2
3	3	2	1	1	1	1
2	2	$\boxed{\mathbf{2}}$	1	0	0	0
1	1	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	$\boxed{\mathbf{1}}$	0	0	1

IsoRegion information

Figure 5: Example of a 2D IsoRegion data structure.

4.2 ISOREGION CONSTRUCTION

Since the IsoRegion information is global in nature, we have to propagate the partial information to the neighboring volumes along the way during the construction process. Here we adopt an iterative approach. In the first iteration the 26 connected neighbors of each voxel are inspected to see if they are homogeneous and take the same voxel value as the center voxel. If so, a '1' is placed there; otherwise, a '0' is stored. In the subsequent iterations we look for larger IsoRegions (i.e. a step-up process) and need not check all the 26-connected neighbors; rather only the eight diagonal-connected neighbors (C_D) since this set of sub-IsoRegions supersedes the others. If all the diagonal-connected neighbors are centers of sub-IsoRegions found in the immediate previous iteration (of dimension parameter ***d***-1), an IsoRegion of dimension parameter ***d*** is found due to the propagation of chess board distance. This iteration step is repeated until no further IsoRegion is found.

Although this processing step takes time, we only need to perform this preprocessing once for each volume data set, and the results can be bundled with the volume material for storage. Note that this preprocessed data structure consumes memory and occupies memory comparable to the data set of the volume. However, current workstations are often equipped with huge amounts of memory, and current hardware capability is enough to cope.

4.3 RAY TRAVERSAL

Before going into the ray traversal scheme, there is another preprocessing step: to pre-compute a look-up a table called the *IsoRegion Step Table* (IRST) for accumulated color and transparency of all possible ray segments. Actually, the IRST is a mapping from voxel value and step value to the accumulated color and transparency of ray segments:

$$\text{IRST}:(\boldsymbol{v}, \boldsymbol{d}) \rightarrow (C_S, B_S)$$

where ***v*** and ***d*** stand for the voxel and step values (the latter being the dimension parameter) of all possible IsoRegions respectively, and C_S and B_S stand for the accumulated color and transparency of the resulting ray segment. The calculation of the IRST is based on the standard illuminance accumulation equation proposed by Levoy [8] with the following initializations:

$$\begin{array}{lllllll} C_i & = & C & = & \text{Color}[\boldsymbol{v}] & \forall\, i \in [1, \boldsymbol{d}] \\ \alpha_i & = & \alpha & = & \text{Opacity}[\boldsymbol{v}] & \forall\, i \in [1, \boldsymbol{d}] \\ \beta_i & = & \beta & = & 1 - \alpha & \forall\, i \in [1, \boldsymbol{d}] \end{array}$$

After simplification we have

$$\text{IRST}(\boldsymbol{v}, \boldsymbol{d}) = (C_S, B_S) = (C(1 - \beta^{\boldsymbol{d}}), \beta^{\boldsymbol{d}})$$

With the IRST we can retrieve the accumulated color and transparency values of a ray segment given the voxel value and the step value.

We use the simple brute-force ray casting algorithm to demonstrate the necessary modifications and show the net speedup from applying the new technique. The caster shoots rays towards each pixel on the image plane from the view point. If the ray misses the volume, the corresponding pixel is shaded black. Otherwise, it pierces the volume and a train of samples is taken along its track. Ordinary ray casters take samples at evenly spaced positions along the ray. However, our algorithm takes samples at unevenly spaced positions depending on the size of the leaps over the IsoRegions encountered.

Figure 6 illustrates how our ray traversal scheme works: Ray A firstly encounters a large IsoRegion with dimension 4 so that it is able to leap over a distance of 4 voxel grid separations before taking the second sample. By looking up the IRST we can accumulate the contribution of 4 voxels at once. The second sample is taken in another IsoRegion. Unfortunately, this time it has no leaping gain since this IsoRegion only has a dimension of 0, which means that no homogeneity coherence can be exploited. Ray C shows the worse case of no gain because a series of IsoRegions of single voxel size is encountered. Although there is no speedup in this case, there is no speed penalty either.

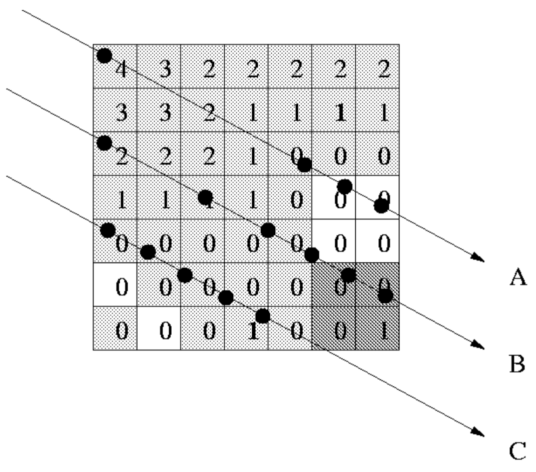


Figure 6: IsoRegion Leaping: unevenly spaced samples depending on IsoRegion dimension (leap value).

The resulting image is identical to the image generated by the brute-force ray casting algorithm, indicating that this acceleration technique is able to preserve its image quality. This follows from the fact that it boosts the ray traversal efficiency by saving unnecessary sample compositions within IsoRegions without affecting the final composited color of the rays. Using the IsoRegion leaping algorithm we can achieve a speedup of 1.7 to 3.0 on average and an error of 0.0%.

5. Distributed Volume Rendering

Volume rendering accelerated with IsoRegion leaping alone still cannot offer an interactive frame rate. To achieve even further speedup we employ distributed volume rendering. Current workstations, even personal computers, are equipped with large amounts of main memory (typically 64 MB or more) and powerful CPUs with continuously dropping prices. Many previous works [7] [9] [10] [11] [17] suggest direct volume rendering is amenable to parallel processing. All these trends together stimulate the development of a loosely-coupled parallel computing environment for direct volume rendering. Likewise, our system executes on a loosely-coupled network of workstations based on various platforms. The linkage consists of a message-passing mechanism on top of TCP/IP.

Figure 7 indicates our simplified system architecture. One machines is assigned to be the console for displaying the rendered images and handling the user interfacing tasks. In addition, the rendering processes are spawned all over the network of participants.

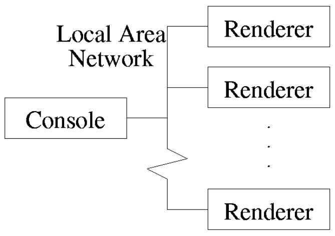


Figure 7 Virtual parallel environment

5.1 DATA AND TASK PARTITIONING

The core of a distributed system is the data and task partitioning scheme. We employ an image space strip-wise task-partitioning scheme as shown in figure 8. On each node we keep the information of the entire volume as well as related information used by the rendering engine. In this way, we avoid massive data communication among the nodes. Although the volume data sets are very large, typically 16 MB or more, the huge memory capacity of current workstations is capable of handling such amounts of data. If we do not replicate the volume data, then some part of the volume or intermediate rendering information has to be exchanged among the renderers using the network. Ma's approach [9] is representative of this technique.

Under our scheme each node is responsible for rendering a strip of the image. We have implemented two perspective ray-casting volume renderers for our distributed rendering system; one is a brute-force caster and the other incorporates IsoRegion Leaping Acceleration [4]. Both have been adapted into our system, and we employed the latter in all the results presented in this paper.

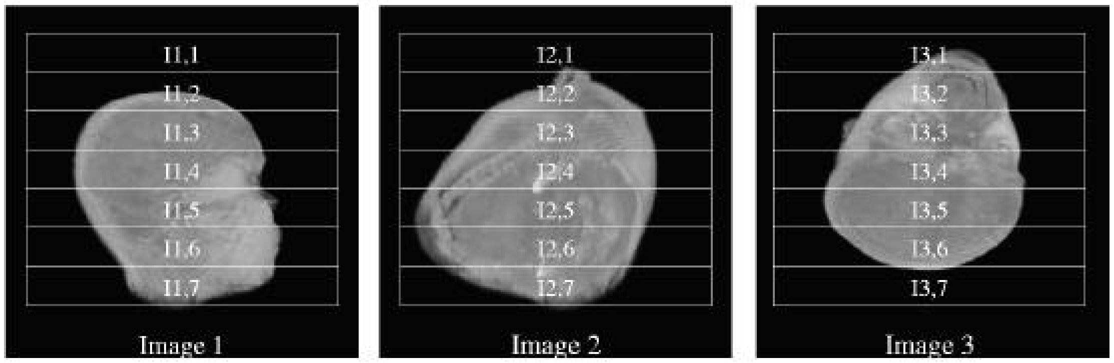


Figure 8 Image based strip-wise partition

Once the image has been chopped into strips, the job of rendering these image strips is distributed among the renderers on different machines. To prevent the renderers from idling, the console machine will not wait to receive the rendered result of the first task before issuing the second task to the same renderer. Thus subtask pre-assignment is employed to further accelerate the frame employed into the results presented later in this paper. Those fast renderers which quickly finish their jobs may work on the image strips in the next frame while slow renderers work on strips in the current frame. The console will display the current frame as soon as all the strips of the current frame are received. Figure 11 shows a snapshot of how a job assignment is done in our distributed rendering system.

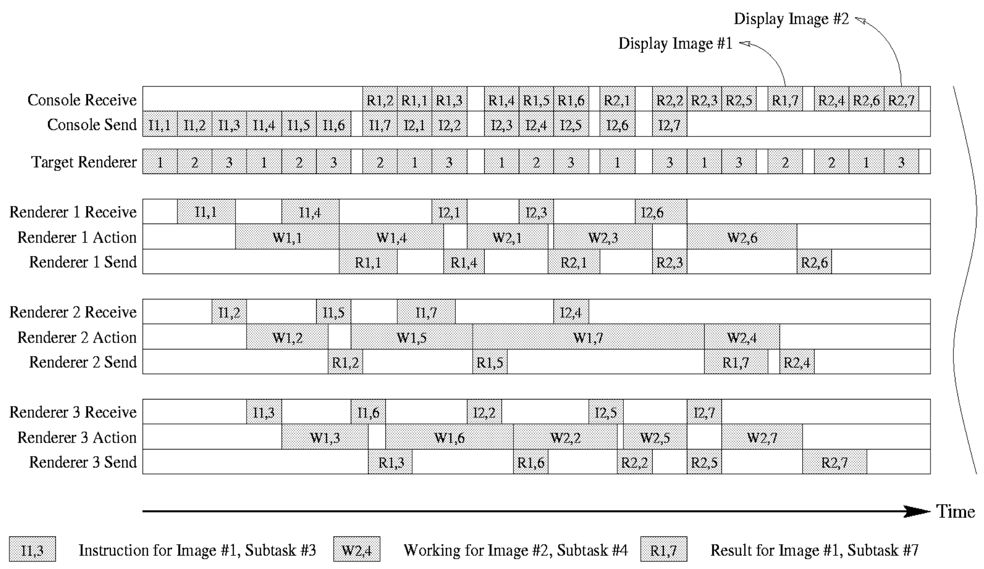
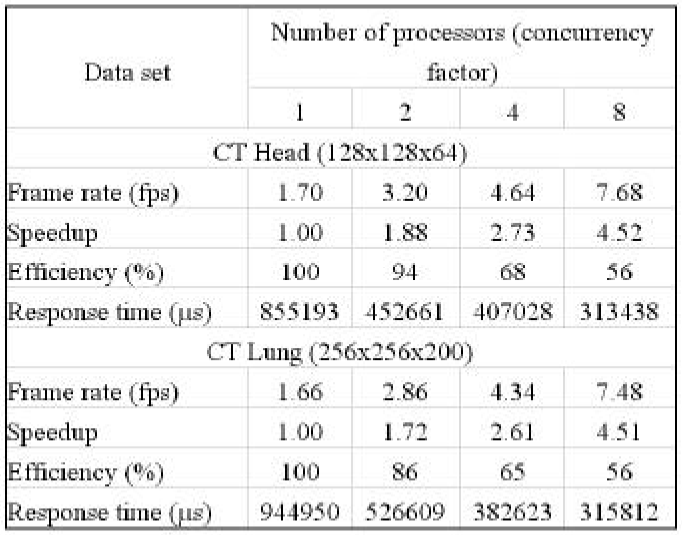


Figure 11 A snapshot of a job assignment in the distributed volume rendering system.

With this scheme the pipeline efficiency is greatly improved because there are less pipeline stalls. In addition, both the console and the renderers are kept busy most of the time. Table 1 reflects these results. Figures 9 and 10 compare frame rate and speedup with varying concurrency factors. Although the ideal case of linear speedup with 100% efficiency is not achieved, our system delivers acceptable performance. With less than 10 Pentium class PC's running Linux in a local area network, an interactive frame rate of about 8 fps is achieved.



Data set	Number of processors (concurrency factor)			
	1	2	4	8
CT Head (128x128x64)				
Frame rate (fps)	1.70	3.20	4.64	7.68
Speedup	1.00	1.88	2.73	4.52
Efficiency (%)	100	94	68	56
Response time (μs)	855193	452661	407028	313438
CT Lung (256x256x200)				
Frame rate (fps)	1.66	2.86	4.34	7.48
Speedup	1.00	1.72	2.61	4.51
Efficiency (%)	100	86	65	56
Response time (μs)	944950	526609	382623	315812

Table 1 Results for subtask assignment scheme with both pre-assignment and multiple image buffers.

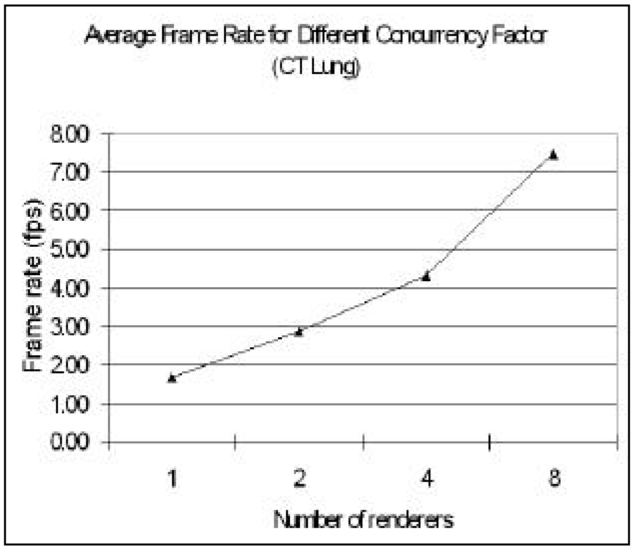


Figure 9 Frame rate.

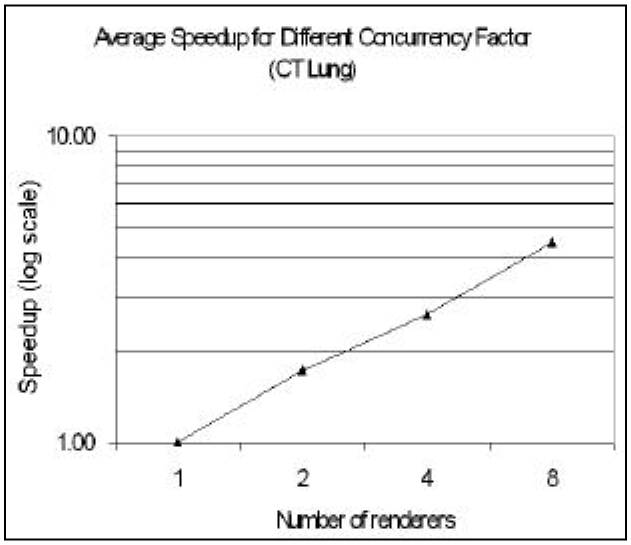


Figure 10 Speedup rate.

5.2 DYNAMIC LOAD BALANCING

On a cluster of heterogeneous loosely-coupled machines, the performance of renderers running on different platforms may vary through time. This may be due to a number of factors:

- Intrinsic computation power deviation of machines
- Background job load
- Variances of network traffic load
- Deviation of the complexities of rendering subtasks

Thus there is a need for balancing the load between different renderers in order to maintain the overall average performance of the system. To make a balanced load, our system adjusts the size of the strips dynamically. During generation of a single frame, we do not have much information on the performance of each renderer. In addition, the effect of load imbalance is not apparent.

When a series of images is to be rendered, the performance estimation of an individual renderer is available by measuring the average time taken by the renderer for its previous subtasks. The console can then assign the various sized subtasks to the renderers accordingly. This load balancing scheme is based on the assumption that the performance of renderers does not change abruptly. This estimation is a valid approximation of performance within a small time interval.

To avoid a sudden change of load (band-band controlled load balancing), it is desirable to smooth out the effect of a single instance of performance disorder. Here we introduce a *forgetting function* for the performance estimation of the renderer *i* at epoch *t* (assuming there are *n* renderers):

$$r_i(t) = \frac{m_i(t)^{-1}}{\sum_{i=1}^{n}(m_i(t)^{-1})}$$
$$p_i(t) = p_i(t-1) * l + r_i(t) * (1-l)$$
$$p_i(0) = n^{-1} \quad \forall i \in [1, n]$$

The parameters $m_i(t)$ and $r_i(t)$ are, respectively, the measured time taken and the relative performance of renderer *i* at epoch *t*. The function $p_i(t)$ is the estimated performance measure of renderer *i* at epoch *t*. It is an accumulated value over time. The damping factor $l \in [0, 1]$ determines the forgetting characteristics of the performance measure. The value of l should be tuned for different systems and configurations. Generally, any value between 0.1 and 0.9 should be fine. Our experiment suggests l=0.9.

We have also performed experiments in a heterogeneous environment. Table 2 shows the results of heterogeneous rendering where several parameters are fixed. The load balancing parameter l is set to 0.9, and there are 16 renderers consisting of a mixture of 2 Sun's, 2 SGI's and 12 PC Pentium's. With this heterogeneous environment, our volume rendering system scores a frame rate of over 10 Hz.

Data set	Frame rate (fps)	Speedup	Eff (%)	Resp. time (µs)
Head	16.91	9.98	62	396048
Lung	15.83	9.54	60	338778

Table 2 Results of heterogeneous rendering.

6. Intelligent Path Tracking

As mentioned before, a restricted movement along the bronchial tree is useful to the doctor who is planning bronchoscopic surgery. To restrict the motion we need to know the exact position of all involved bronchial tubes in 3D. To track the bronchial tree, we propose a new tracking algorithm and develop an intelligent path tracking sub-system based on this algorithm. Traditionally, tracking is done by hand, although this is a labor-intensive and time-consuming process. On the other hand, using a fully automatic algorithm may not be reliable. For these reasons, we believe a semi-automatic algorithm is more practical; hence we propose the semi-automatic 3D intelligent path tracker.

Mortensen and Barrett [13] proposed a method called “intelligent scissors” to segment the 2D image in a semi-automatic manner. Each pixel in a 2D image is regarded as a node in a graph. Neighboring nodes are connected by undirected edges, and each edge is assigned a cost in accordance to specified user defined functions. Therefore the problem of finding a path between two pixels becomes a problem of finding the path with minimum cost. This cost-minimized path can be found using Dijkstra’s algorithm method [2] that has been proved useful in finding boundaries in 2D images. In fact, tracking the bronchial tube in the volume data section is actually a problem of finding boundaries in a 3D volume. This motivates us to use an intelligent scissor-like method to track the tubular structure in 3D.

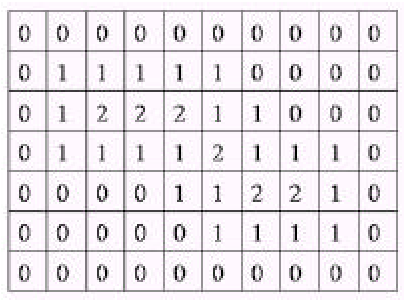
The extension from 2D to 3D is straightforward. Similar to the pixel graph depicted in 2D image, each voxel (of a volume) is now a node, and every node is connected to 26 neighboring nodes (voxels) by 26 edges. Note that the speed of finding a 3D path is directly related to the number of nodes and edges in the graph. The algorithm is very similar to its 2D counterpart except the number of nodes and edges is tremendously increased.

6.1 3D GRAPH CONSTRUCTION WITH ISOREGION

Since there is a huge number of nodes and edges produced, several tens of seconds can be taken to extract just one path from a graph with 1 million nodes and about 13 million edges, even with the aid of a powerful computer. This time required for extraction is due to Dijkstra’s algorithm, which has a running time of $O(|E|+|V|\log|V|)$, where the Fibonacci heap [3] is used to implement the priority queue, with V being defined as the set of nodes and E is the set of edges.

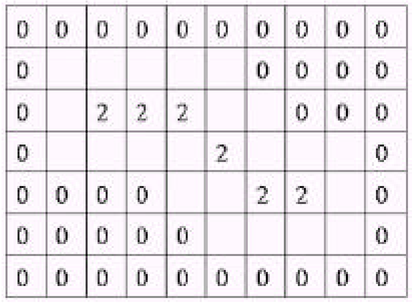
Hence the problem becomes not only how to reduce the number of voxels, but how to preserve the connectivities of distinct objects inside the volume. A voxel with zero gradient means its value is the same as the neighboring voxels. However, we cannot naively eliminate all voxels which have zero gradient. Instead, we should keep some of them and use those voxels to connect them between different groups. To solve the problem we use the IsoRegion [4] described previously. All voxels inside an IsoRegion with zero gradient are cut away except the innermost voxels. Connectivity is then established by linking some of the boundary voxels of an IsoRegion to these innermost voxels. However, linking all boundary voxels to the center can produce too many edges if the size of the IsoRegion is too large.

To facilitate the discussion we shall illustrate the idea using 2D examples from now on. Figure 12(a) shows the 2D IsoRegion map of a 2D version (image) of a volume data set. As you can see, all voxels inside an IsoRegion are pruned away, except the boundary voxels and the center voxel. Higher priority is given to larger IsoRegions. For example, the voxels in Figure 12 marked "1" correspond to Isoregions of dimension one. However, they are still cut away, as shown in Figure 12(b), since they are contained in larger IsoRegions of dimension two. Finally, the graph can be completed by linking the boundary voxels to the center voxels as shown in Figure 12(c). Usually not all boundary voxels of an IsoRegion are left after pruning because they are contained in other IsoRegions, and links are added to those remaining boundary voxels.



0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	1	2	2	2	1	1	0	0	0
0	1	1	1	1	2	1	1	1	0
0	0	0	0	1	1	2	2	1	0
0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

Figure 12(a): The IsoRegion map



0	0	0	0	0	0	0	0	0	0
0						0	0	0	0
0		2	2	2			0	0	0
0					2				0
0	0	0	0			2	2		0
0	0	0	0	0					0
0	0	0	0	0	0	0	0	0	0

Figure 12(b): After node pruning

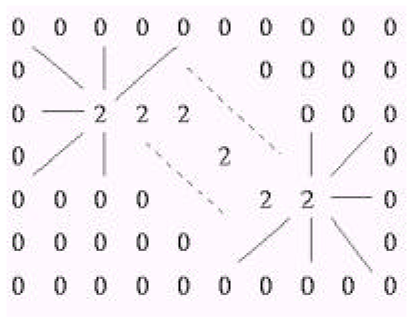


Figure 12(c): Linking

6.2 COST FUNCTION

In Dijkstra's algorithm the cost function is directly related to the final result. A bad cost assignment can either make the result far away from the desired one or make it too sensitive to noise. Mortensen and Barrett's 2D algorithm [13] proposed the following cost function which is composed of three functions: Laplacian zero-crossing (f_Z), gradient magnitude (f_G), and gradient direction (f_D). The cost of an edge between node *u* and node *v* is defined as

$$cost(u,v) = w_Z \times f_Z(v) + w_D \times f_D(u,v) + w_G \times f_G(v)$$

where w_Z, w_D and w_G are weights for the functions.

However, their cost function is specially designed for 2D image segmentation, and for this reason it does not suit our purpose. Instead, we define the cost function as follows: Let f_g and f_z be the 3D gradient and the 3D Laplacian zero-crossing functions respectively. If *p(u)* is the position vector of *u*, then the cost function is

$$cost(u,v) = ||p(u) - p(v)|| \, (w_g \times f_g(u,v) + w_z \times f_z(u,v))$$

where w_g and w_z are the weighting factors controlling the 3D gradient and 3D Laplacian zero-crossing functions respectively. Experiments show that w_g=0.7 and w_z=0.3 give acceptable results.

If (I_x, I_y, I_z) is the gradient vector of a voxel *u*, the gradient function of an edge can be represented by

$$f_g(u,v) = 1 - \frac{G(u)+G(v)}{\max(G)}$$

where

$$G(u) = \sqrt{I_x^2(u) + I_y^2(u) + I_z^2(u)}$$

Finally, the equation of Laplacian zero-crossing function is given by

$$f_z(u,v) = \begin{cases} 0 \cdots if & \nabla^2 V(u) * \nabla^2 V(v) \le 0 \\ 1 \cdots if & \nabla^2 V(u) * \nabla^2 V(v) > 0 \end{cases}$$

where

$$\nabla^2 V(u) = \left\{ \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} \right\}$$

and it can be approximated by the following discrete form:

$$\nabla^2 V(x,y,z) \approx \frac{\left(27V(x,y,z) - \sum_{i=-1}^{1}\sum_{j=-1}^{1}\sum_{k=-1}^{1} V(x+i,y+j,z+k)\right)}{26}$$

6.3 FINDING THE COST-MINIMIZED PATH

Normally Dijkstra's algorithm originates from the source node and stops searching if the target node is reached. That means the graph is only partially filled if a "point to point" shortest path is to be found. Therefore, if the source node is fixed, the partially filled graph can be re-used in order to save time, since some or most of the nodes on the path have already been traversed. This property allows us to incrementally construct the bronchial tree structure since all branches of the tree have a common source/root node. The building of a bronchial tree can be done by a simple algorithm below:

```
Algorithm 1 ConstructTree
1   r = user specified root node
2   T = {user specified target nodes}
    /* V={all nodes}, E={all edges} */
3   G = graph(V, E)
4   B(u) = nil ∀ u ∈ V
5   for each u ∈ T
6       if B(u) = nil
7       then Dijkstra(G, B, r, u)
```

As a result, the path information is stored in the node array B(*u*), which stores the back pointers that indicate the paths (see Figure 13(a)). Therefore, a path from root node *r* to a target node *v* can be found by traversing the back pointers and the path can be represented as

path = {v, B(v), B(B(v)), B(B(B(v))), ... , r}

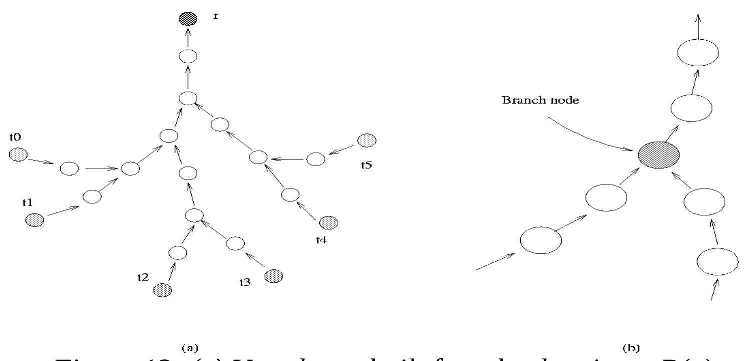


Figure 13: (a) Vessel tree built from back pointersB(u)
(b) Branch node is pointed by two or more other nodes.

Figure 14 shows two screen shots of tracking the lung bronchial tree using the proposed method. We have implemented it on the *Virtual Workbench* [16], a 3D virtual environment. The 3D position of the starting and the ending points of the airway can be naturally specified in this 3D virtual environment using a 3D input device; provided that the user specifies the root node and the target nodes correctly, say, all within the airway. The extracted tree structure will then accurately reflect the branching structure of the bronchial vessels. This is guaranteed because the wall of the airway has such a high cost that the cost minimization process always prevents wall penetration.

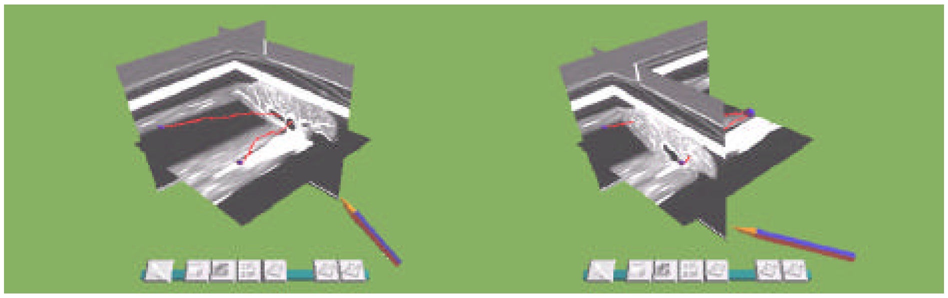


Figure 14 Two screen shots of tracking the airway in the lung volume.

7. Conclusions

We have developed a virtual bronchoscopy system which simulates the real bronchoscopy procedure conducted on a human patient. We feel that the benefits of this technique reach into the domains of enhancing the quality of life of the patient as well as giving the doctor a useful tool that can make his/her job easier and the diagnosis more accurate. The most important facet of this methodology is that doctor can now diagnose and plan preliminary strategies for the patient for real bronchoscopic surgery without jeopardizing the patient's health.

The fundamental components of this system are composed of two sub-systems: the interactive visualization and navigation sub-system, and an intelligent path tracking sub-system. To achieve an interactive frame rate, we have employed IsoRegion leaping and distributed volume rendering on a network of low-cost computers. The final frame rate achieved is over 10 Hz. In the context of bronchoscopy simulation, a restricted motion along the bronchial tube is important and useful. The bronchial tree structure is tracked in our intelligent path tracking sub-system with our proposed semi-automatic and reliable technique called the Intelligent Path Tracker.

In the future we plan to extend the system to include a force feedback feature as well as an input device which will be able to mimic the real bronchoscope. The current input device we are using consists of a 3D pen, which can be even further developed in an effort to meet the needs of the doctor conducting the bronchoscopy. We believe that making these improvements will further enhance the realism of the procedure and diminish the differences between our system and conducting a real bronchoscopy.

8. Acknowledgements

This work is supported by RGC Earmarked Grant CUHK4162/97E of Hong Kong.system and conducting a real bronchoscopy.

REFERENCES

[1] D. Cohen and Z. Sheffer, "Proximity clouds - an acceleration technique for 3D grid traversal." *The Visual Computer*, 11(1):27-38, November 1994.

[2] E. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, 1:269-270, 1959.

[3] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms." *Journal of the ACM*, 34:596-615, 1987.

[4] P. F. Fung and P. A. Heng, "Efficient volume rendering by isoregion leaping acceleration." In *Proceedings of The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98*, volume 3, pages 495-501, Feburary 1998.

[5] B. Geiger and R. Kikinis, "Simulation of endoscopy." In *Computer Vision, Virtual Reality and Robotics in Medicine*, pages 277-281, 1995.

[6] L. Hong, A. Kaufman, Y. C. Wei, A. Viswambharan, M. Wax, and Z. Liang, "3D virtual colonoscopy." In *Proceedings 1995 Biomedical Visualization*, pages 26-32, October 1995.

[7] P. Lacroute, "Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization." In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 15-22, 1995.

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

[9] K. L. Ma and J. S. Painter, "Parallel volume visualization on workstations." *Computer and Graphics*, 17(1):31-37, 1993.

[10] R. Machiraju, L. Schwiebert, and R. Yagel, "Parallel algorithms for volume rendering." Technical report, the Ohio State University Department of Computer and Information Science, October 1992.

[11] R. Machiraju and R. Yagel, "Efficient feed-forward volume rendering techniques for vector and parallel processors." In *Proceedings of Supercomputing*, pages 699-708, April 1993.

[12] E. McVeigh, M. Guttman, E. Poon, P. Chandrasekhar, C. Moore, E. Zerhouni, M. Solaiyappan, and P. A. Heng, "Visualization and analysis of functional cardiac MRI data." In *Medical Image 94, SPIE conference Proceedings*, volume 2168, Feburary 1994.

[13] E. N. Mortensen and W. A. Barrett, "Intelligent scissors for image composition." In *Computer Graphics (SIGGRAPH'95 Conference Proceedings),* pages 191-198, August 1995.

[14] D. R. Ney et al., "Three-dimensional CT-volumetric reconstruction and display of the bronchial tree." *Investigative Radiology*, 25(6):736-742, January 1990.

[15] Polhemus, *Polhemus 3 Space Tracker System User Reference Manual*. Polhemus Incorporated (A Kaiser Aerospace & Electronic Company), P.O. Box 560, Colchester, Vermont 05446, U.S.A., revision F edition, November 1993. OPM3609-002C.

[16] T. Poston, L. Serra, M. Solaiyapan, and P. A. Heng, “The graphics demands of virtual medicine.” *Computer and Graphics*, 20(1), January 1996.

[17] P. Schröder and G. Stoll, “Data parallel volume rendering as line drawing.” In *1992 Workshop on Volume Visualization Proceedings*, pages 25-32, October 1992.

[18] L. Serra, T. Poston, H. Ng, P. A. Heng, and B. C. Chua, “Virtual space editing of tagged MRI heart data.” In *Proceedings of the First International Conference on Computer Vision Virtual Reality and Robotics in Medicine*, April 1995.

[19] L. Serra, T. Poston, W. L. Nowinski, B. C. Chua, H. Ng, and P. K.Pillay, “The brain bench planner and trainer for minimal access surgery.” In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 191-192, July 1996.

[20] M. Solaiyappan, T. Poston, P. A. Heng, E. R. McVeigh, M. A. Guttman, and E. A. Zerhouni, “Interactive visualization for rapid noninvasive cardiac assessment.” *IEEE Computer*, pages 55-62, January 1996.

[21] M. Šrámek, “Fast ray-tracing of rectilinear volume data.” In *Virtual Environments and Scientific Visualization ’96, Proceedings of the Eurographics Workshops*, pages 201-210, 1996.

[22] R. Yagel and Z. Shi, “Accelerating volume animation by space-leaping.” In *Proceedings of Visualization ’93*, pages 62-69, 1993.

[23] K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever, “Acceleration of ray-casting using 3D distance transforms.” In *Visualization in Biomedical Computing II, Proc. SPIE 1808*, pages 324-335, 1992.

BIOGRAPHIES

Pheng-Ann Heng is an associate professor at the Department of Computer Science and Engineering at the Chinese University of Hong Kong. He received a B.Sc. in Computer Science from the National University of Singapore, an M.Sc. in Computer Science, an M.A. in Mathematics, and a Ph.D. in Computer Science from Indiana University, U.S.A. He worked as a Research Scientist at the Institute of Systems Science of the National University of Singapore before he joined the Chinese University of Hong Kong in 1995. He was a visiting scientist at The Johns Hopkins Medical School during 1993 and 1994. He is currently the director of the Virtual Reality, Visualization, and Imaging Research Center at CUHK and leads several research projects in developing virtual environments for surgical simulation and realtime scientific visualization. His current research interests include virtual reality applications in medicine, interactive scientific visualization, 3D medical imaging, 3D user interfaces, and computer graphics.

Contact information:

Dr. Pheng-Ann Heng
Dept of CSE
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong
Phone: +852-26098424

Fax: +852-26035024
Email: mailto:pheng@cse.cuhk.edu.hk)

Ping-Fu Fung received B.Sc. and M.Phil. degrees in Computer Science from the Chinese University of Hong Kong in 1996 and 1998, respectively. After his graduation, he has been working as a research assistant at the Virtual Reality, Visualization, and Imaging Research Center at the Chinese University of Hong Kong. His current research interests include computer applications in medicine, volume visualization, and 3D user interfaces.

Contact information:

Mr. Ping-Fu Fung
Department of CSE
The Chinese University of Hong Kong
Phone: +852-26098427
Fax: +852-26035024
Email: mailto:pffung@cse.cuhk.edu.hk

Tien-Tsin Wong is a visiting assistant professor in the Computer Science Department at Hong Kong University of Science & Technology since August 1998. He graduated from the Chinese University of Hong Kong in 1992 with a B.Sc. degree in Computer Science. He received M.Phil. and Ph.D. degrees in Computer Science from the same university in 1994 and 1998, respectively. His research interests include image-based rendering, photorealistic rendering, medical visualization, and natural phenomena modeling.

Contact information:

Dr. Tien-Tsin Wong
Computer Science Dept
Hong Kong University of Science & Technology
Clear Water Bay, Hong Kong.
Phone: +852-23587021
Fax: +852-23581477
Email: mailto:ttwong@acm.org

Kwong-Sak Leung is professor and chairman of the Department of Computer Science & Engineering at the Chinese University of Hong Kong. He received B.Sc. and Ph.D. degrees from the University of London in 1977 and 1980, respectively, and is a member of IEE and ACM, a senior member of IEEE, and a chartered engineer. He is member of Editorial Board for Fuzzy Sets and Systems and the IT Magazine (1994-98). He has served as chairman and member of numerous international conference organizing and program committees. He has authored or co-authored over 120 publications. His research interests are in the areas of knowledge engineering, expert systems and data mining, genetic algorithms and programming, automatic knowledge acquisition, Chinese processing, fuzzy logic applications, and AI architecture.

Contact information:

Dr. Kwong-Sak Leung
Department of CSE
The Chinese University of Hong Kong
Phone: +852-26098408
Fax: +852-26035024
Email: mailto:ksleung@cse.cuhk.edu.hk

Han-Qiu Sun is an assistant professor at the Department of Computer Science and Engineering at the Chinese University of Hong Kong since August 1996. She received a B.Sc. in Electrical Engineering from Huazhong University of Science and Technology in China, an M.S. in Electrical Engineering from the University of British Columbia, and a Ph.D. in Computer Science from University of Alberta in Canada. She has published more than 50 refereed technical papers in *The Journal of Virtual Reality*, *Computers & Graphics*, *Visualization and Computer Animation, IEEE Transactions on Systems, Man, And Cybernetics*, *IEEE Journal of Computer Graphics and Applications*, and has refereed book chapters and international conference papers. Her current research interests include VR, interactive behavioral animation, hypermedia, computer-assisted surgery, robotics simulation, internet-based visualization and navigation.

Contact information:

Dr. Han-Qiu Sun
Department of CSE
The Chinese University of Hong Kong
Phone: +852-26098399
Fax: +852-26035024
Email: mailto:hanqiu@cse.cuhk.edu.hk