Virtual Colonoscopy Powered by VolumePro

Wei Li[1], Ming Wan[1], Baoquan Chen[1] and Arie Kaufman[1]

Department of Computer Science and Center for Visual Computing (CVC)
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

Abstract

Virtual colonoscopy is a non-invasive method for detecting polyps inside the colon. We developed a virtual colonoscopy system on a high-end 16-processor SGI Challenge with an expensive hardware graphics accelerator. To increase its availability for mass screening, we ported the system to a low cost PC. Since interactivity is critical in such an application, we took the advantage of the recently released VolumePro, a volume-rendering PC board, that can render a 256^3 volume in real time. However, VolumePro currently does not support perspective projection, which is essential to interior colon navigation. Besides, the patient colon data is usually much larger than 256^3 and cannot be rendered in real time.

In this paper, we present a new system design that tightly combines sub-volume rendering and slabbing for perspective approximation. We also propose to use image-warping to reduce the parallel-to-perspective distortion. The average rendering speed is about 30 frames per second, and varies from 20 to 40 depending on the camera position and orientation. Moreover, our rendering with sub-volumes can be naturally extended to an image-based rendering scheme that shows a similar performance but does not need VolumePro for the actual colon navigation.

Keywords: Virtual colonoscopy, virtual endoscopy, scientific visualization, volume rendering, interactive navigation, hardware acceleration, perspective projection.

1 Introduction

Colorectal carcinoma is the third most commonly diagnosed cancer and the second leading cause of death from cancer in the United States. Often it is diagnosed at an advanced stage, after the patient has developed symptoms, explaining its high mortality rate [1]. Since most cancers arise from polyps over a 5 to 15 year period of malignant transformation, screening programs to detect small polyps less than 1 cm in diameter have been advocated. Optical colonoscopy is the commonly used diagnostic procedure. Unfortunately most people do not follow this recommendation because of the associated risk, discomfort, and high cost. In order to dramatically increase the number of people willing to participate in screening programs, efforts have been made towards a computer-based screening modality, called 3D virtual colonoscopy, as an alternative to optical colonoscopy, by employing advanced computer graphics and visualization techniques [2, 3, 4].

The virtual colonoscopy system takes a spiral CT scan of the patient's abdomen after the entire colon is cleansed and distended with room air or CO_2. Several hundred high-resolution CT images are rapidly acquired during a single breath-hold of about 30-40 seconds, forming a volumetric abdomen data set. A model of the real colon is then segmented from the abdomen data set. It can be viewed either by automatic planned navigation following the center-line of the lumen of the colon, providing a general overview of the inner colonic surface, or by interactive navigation for a more detailed study of suspicious regions. In our previous work, we developed such a virtual colonoscopy system on high end 16-processor SGI Power Challenge with expensive Infinite Reality graphics hardware acceleration [4, 5, 6, 7]. We have already confirmed that we can visualize polyps as small as 3 mm, and polyps that have been detected in optical

[1]Email:{liwei, mwan, baoquan, ari}@cs.sunysb.edu

colonoscopy have also been identified with our virtual colonoscopy. Our work in this paper is to make our technology easily accessible on a low cost PC (Personal Computer) so as to increase the availability as a mass screening procedure.

Real-time rendering rates of 10-30 frames per second are critical for an interactive virtual navigation and detection inside the virtual human colon. We previously proposed and implemented two different real-time colonic rendering techniques on high-end SGI workstations, respectively using fast surface rendering [4, 5] and direct volume rendering techniques[6, 7]. Fast surface rendering time was achieved by exploiting the Infinite Reality graphics hardware accelerator on SGI workstations to display those colon surfaces visible to the current view. The colon surface was extracted from the colon volumetric data in a preprocessing stage. Fast direct volume rendering rates were reached by using 16 processors to directly render the colon images from the original 3D volumetric colon data. Our preliminary experimental results indicated that the direct volume rendering technique provided more realistic colonic images, more flexible visualization of interior structures for polyps and other abnormalities, and shorter preprocessing time. Physicians have confirmed that our direct volume rendering images of the human colon are very close to what they observed in optical colonoscopy. Consequently, when our virtual colonoscopy system is ported and implemented on the low-end PC, it is critical to maintain the high image quality and rendering speed of our direct volume rendering techniques.

Mitsubishi Electric has developed a volume-rendering PC board, called VolumePro, which supports real-time direct volume rendering rates. Currently, the first generation VolumePro board from Mitsubishi [9, 10], which is based on the Cube-4 architecture developed at SUNY at Stony Brook [11], contains a vg500 chip and 128 MB of SDRAM. Note that the vg500 chip can render 500 million samples per second in parallel projection. If one sample is taken per voxel (either 1 byte or 2 bytes) so that no data is missed, then the vg500 chip can render a 256^3 volume at 30 frames per second.

While implementing the virtual colonoscopy on VolumePro, we are facing two major difficulties. First, the current version of VolumePro only supports parallel rendering, whereas perspective rendering is critical to the virtual navigation inside of a colon. In a preliminary version of the PC-based virtual colonoscopy [8], we approximated the perspective volume rendering by parallel rendering slabs and blending slabs perspectively. Adjusting the thickness of each slab gives a tradeoff between image quality and rendering speed. Second, the current version of VolumePro is still not fast enough to render the whole colon dataset (typically 512^3) in real time. Slabbing the volume to approximate perspective rendering makes the situation even worse, since VolumePro needs to render multiple times for a single frame. To reduce the rendering time, we used a set of manually created sub-volumes. Those sub-volumes are overlapped with their neighbors to ensure there is only one sub-volume containing all the visible voxels for any given camera parameters. The system was demonstrated at the Radiological Society of North America (RSNA) in Chicago, November 1999. Although both the image quality and rendering performance are considered acceptable, the image is sort of blurred compared with a software ray-caster, such as the one we implemented on SGI workstations [6, 7], and the rendering speed is still less than 10 frames/s (with an average of 6 frames/s for the whole navigation, and the frame rates varies from 3 to 9 frames) even without super sampling and with a modest number of slabs for perspective effect. The blurring is mainly because the resolution of the image generated by VolumePro is comparable to the number of voxels processed. While navigating the interior of the colon, it usually only needs a small subset of voxels. Hence zooming the image to fit the view is unavoidable. VolumePro provides super- sampling to cope with this blurring, but double the image resolution in any of the dimensions more than doubles the rendering time.

In this paper, we propose a new rendering algorithm for VolumePro that seamlessly combines slab-based perspective rendering and sub-volume rendering for acceleration. In our method, we create three sets of subvolumes (volume-aligned slabs), one for each major axis. For a particular camera position and orientation, the intersection of a few slabs and the view frustum, which is (almost) the smallest possible subset of voxels, is sent to VolumePro for rendering. The method is similar to the idea of using a sequence of connected boxes to encapsulate the colon and rendering only a set of subvolumes specified by a few boxes [12]. But our sub-volumes are tighter bounding boxes of slabs for perspective rendering and our subvolumes are easier to create. Taking advantage of the opaqueness of the colon wall, we introduce an image-warping method that reduces, if not completely eliminates, the distortion caused by simulating perspective rendering with parallel rendering. Since the shape of our sub-volumes is much more regular,

we can reuse slab images for consecutive frames to further accelerate the rendering speed. To increase the reusability of slab images, our system predicts the next one or more positions of the camera, and adjusts the slabs accordingly. We also implement a software-controlled pipeline that exploits the parallelization among CPU, VolumePro board and an ordinary graphics card to hide the overhead such as preparing slabs and blending slab images. Finally, we point out that our method can be naturally extended to an image-based rendering scheme that does not need VolumePro board but can exhibit similar performance. Figure 1 is a screen shot of the system. Although we focus on virtual colonoscopy, the techniques we propose in the paper also apply to virtual endoscopy or general tubular structures.

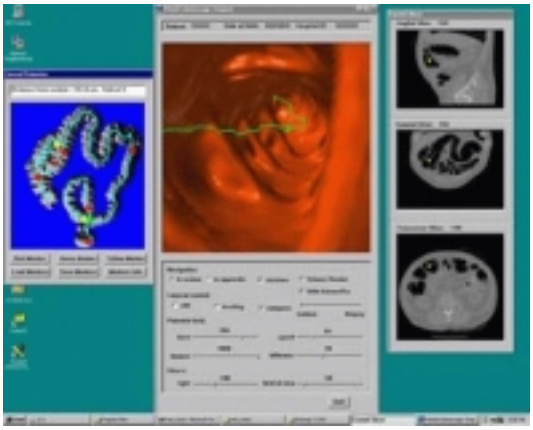

Figure 1: *A screen shot of the virtual colonoscopy system*

2 Perspective Projection based on Slabbing and Image-Warping

2.1 Parallel-to-Perspective Distortion

The current version of VolumePro supports only parallel projections. Yet the navigation inside tubular objects, like a colon, definitely requires perspective projections, since parallel projection compresses a straight tube into a 2D ring, while perspective projection creates a view, which allows the user to see the interior walls of the tube.

In our previous work, we used a slabbing technique in which the volume is partitioned into slabs of slices aligned either along a major volume axis or parallel to the image plane [8, 12]. Figure 2 illustrates this process using three axis-aligned slabs. Each slab image is texture mapped to a surface polygon, which is parallel to the slab face most perpendicular to the view direction and dichotomizes the slab. Then, those images are alpha blended from back to front perspectively.

As in Figure 3, the perspective rendering algorithm actually approximates a ray with a set of disjoint line segments that are generally non-collinear. The error of such approximation depends on the angle of the casted ray and the thickness of slabs. We can trade off between quality (with a large number of thin slabs) and speed (with a small number of thick slabs) by choosing slab thickness.

For an application like virtual colonoscopy, the slab surfaces that are most perpendicular to the viewing direction, which should be invisible in real perspective rendering, display the most notable distortion, as shown in Figure 4. Figure 4(a) is an actual image from a colon navigation. Figure 4(b) illustrates such distortion by simplifying the colon with three pipes segments.

Since perspective distortion shrinks objects in the distance, it also shrinks the errors in the distance. This allows us to use the principles of adaptive perspective volume rendering [13] to adaptively modify the slab thickness along the viewing direction with constant error bounds. We derived that the slab thickness should be a geometry series so as to distribute the error uniformly on the image plane, hence making the distortion less noticeable [8].

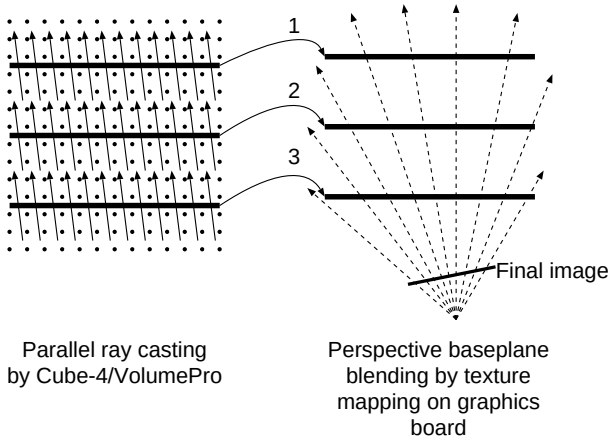

Figure 2: *Approximation of perspective volume projection by rendering slabs using parallel volume rendering hardware and blending the slab images perspectively.*

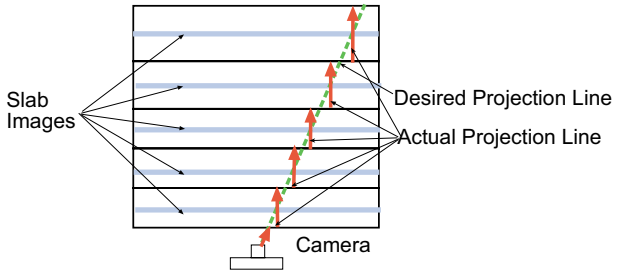

Figure 3: *Distortion of approximating perspective volume projection by perspectively blending parallelly rendered slab images.*

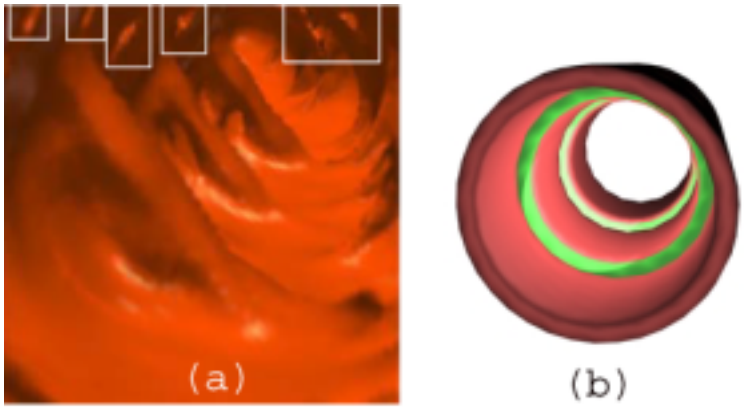

Figure 4: *Images showing the distortion of the approximated perspective volume rendering. (a) is an actual image from a colon navigation. Highlighted areas marked with white rectangles exhibit notable distortion. (b) is the image of a three-segment pipe. The internal ring is the face of the back segment uncovered by the front segment.*

2.2 Reducing Parallel-to-Perspective Distortion by Image Warping

Both of the above methods [13, 8] require the slabs to have variant thickness, which poses difficulty in reusing the slab images, an essential technique for accelerating the rendering process. If we choose slabs

of a constant thickness, the thickness of each slab should equals to the thickness of the closest slab of the variable thickness method, which demands more slabs to be rendered.

Actually, the rendering of the interior wall of the colon is intrinsic to the following features:

· The camera is always inside the colon during the virtual navigation [4].

· The colon wall is opaque except when the user is inspecting abnormal areas in the virtual biopsy mode and the navigation is suspended [7].

· Any short-enough segment of the colon can be approximated by a segment of a tube.

Based on the above observations, we propose to use a geometry-assisted image-warping to reduce the most significant distortion as discussed in the previous section (See Figure 3). We approximate the colon with a set of pipe segments of rectangular cross sections. Then, the slab we use for perspective rendering is a sub-volume containing a pipe segment. Each slab has a hole with two openings, called portals. Portals are used to determine visibility [5]. Figure 5 displays in 2D the geometry of such a slab. Figure 6 is the 3D rendering of a slab with a rectangular hole in it. The image of the slab is mapped to the interior surface of the pipe depicted as thick lines in Figure 5 rather than mapped to a flat surface. Since the camera is never outside the colon, the part of the view volume between the two faces of any segment that are most perpendicular to the viewing direction is always inside the segment. Therefore, the profile of the projection of any slab is a quadrilateral rather than a hexagon. We also construct the portal in such a way that the two portals are on opposite faces of a slab. Figure 7 illustrates the fact that mapping a slab image to the model of a pipe segment is equivalent to enlarging the projection of front portal and shrinking that of the back portal in 2D.

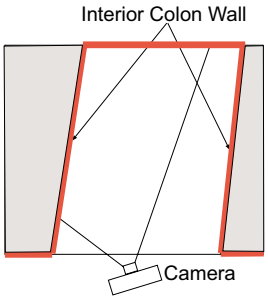

Figure 5: *Geometric model of a pipe segment.*

Figure 6: *The 3D rendering of a pipe segment.*

Intuitively, it is not difficult to imagine that enlarging and shrinking slab images around portals will significantly reduce the distortions presented in Figure 3. With this technique, we can choose thicker and fewer slabs and expect similar or better image quality as the method without such warping. With the help of image-warping, we choose to use slabs of uniform thickness.

Mueller et al. used depth information to enhance their IBR-Assisted volume rendering [14]. The difference between their method and ours is that our model is simpler since it is specifically designed for tubular shaped- object, like colon. Besides, rendering assisted with our model is less likely to generate gaps between slab images when slabs are reused and viewed from different angles as discussed in [14].

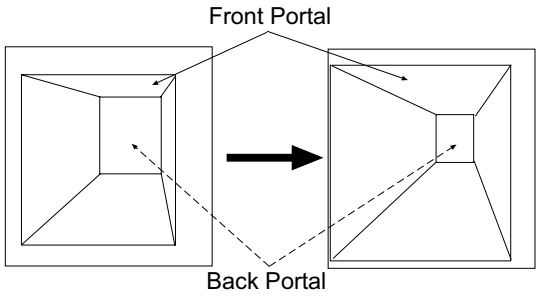

Figure 7: *Image warping assisted with the model of a pipe segment. On its 2D projection, the front portal is enlarged while the back portal is shrank.*

3 Rendering with Slabs

As mentioned in the previous section, the purpose of slabs is not only to achieve perspective effect, but also to process the least voxels. For each frame image, the union of the slabs used is the smallest bounding box of the intersection of the view-volume and the visible interior of the colon. The slabs are dependent on the parameters of the camera, therefore, should be found on the fly. To accelerate the slabbing process, as well as to simplify the visibility detection as in [5, 12], we specify a series of boxes during the preprocessing, whose union covers all the interior voxels of the colon.

3.1 Image-Aligned Slabbing vs. Volume-Aligned Slabbing

To approximate perspective projection with VolumePro, we have two choices: image-aligned slabbing and volume-aligned slabbing. In [8], we render a set of image-aligned slabs for its intuitiveness, since the hexagon geometry returned by VolumePro that is mapped with the slab image is always parallel to the image plane. Moreover, sub-volumes should be clipped by the near plane that is also parallel to the image plane. Image-aligned slabbing naturally contains this step.

In this paper, we would rather argue for the use of volume-aligned slabs. First, this makes the number of possible boxes limited to a small number, since we only need to consider boxes aligned to the axis of the whole volume dataset. Second, some claim that an image generated from image-aligned slabs has a better quality than that from volume-aligned slabs. This is not true at least in the VolumePro case. VolumePro casts the rays onto an intermediate viewing surface called a base plane that is parallel to and co-planar with the volume surface that is most nearly parallel to the image plane and closest to the eye. Then, the image on the baseplane is warped to the image.

3.2 Creating Slabs

The current version of VolumePro is supposed to allow the user to specify an active sub-volume and then only process the voxels contained in that sub-volume. A limit to an active sub-volume is that all six of its faces should be on 32-voxel boundaries.

We can also use the CPU to chop the whole volume dataset, and create a sub-volume for VolumePro. The advantage of the software-created sub-volume is that the border of the sub-volume does not need to be a multiple of 32 voxels in the original volume on any axis, hence leaving more freedom in designing the system. Of course, VolumePro will expand the sub-volume if necessary to make its size multiple of 32 bytes on all three dimensions. A disadvantage obviously is that it takes longer to create a sub-volume with CPU and to load it into the voxel memory than calling the ActiveSubVolume function of VolumePro. However by parallelizing the operations of CPU (discussed below), VolumePro and a graphics card, the chopping can be hidden in the time required by VolumePro to render.

Another choice is to create and load the slabs at the preprocess stage. It is feasible even if we have to duplicate storage, since the current version of VolumePro is equipped with 128MB memory and the voxels inside the colon only accounts for a small percentage (about 17%) of the whole dataset. Apparently, the

precreated slabs are view-independent. Hence, we can not utilize view-frustum clipping to reduce the size of slabs, as will be discussed in the following section.

3.3 Boxing

Before the actual navigation, we calculate three sets of volume-aligned boxes that separate voxels inside the colon from the others. There is one set associated with each of the major axes of the volume dataset. A major axis is perpendicular to the largest two faces of a slab associated with it[2]. Any box is slab-shaped and its size along its associated volume axis is exactly the same as the slab thickness for perspective rendering. The union of each set of boxes enclosed all voxels necessary for the navigation inside the colon. It also includes some extra voxels that are not inside the colon, since any face of a box is a rectangle. Figure 8 shows the boxes associated with the Y axis. We refer to the axis as the *facing direction* of the corresponding boxes. Therefore, in Figure 8, the facing direction of all the boxes is parallel to the Y axis. The thick lines are the projection of the portals of the boxes.

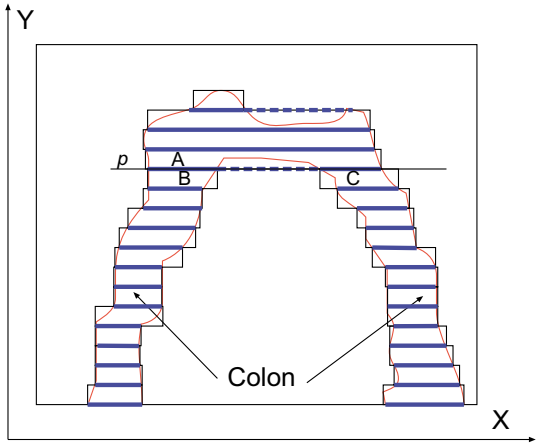

Figure 8: *Boxes for the Y axis.*

3.3.1 Construction of the boxes

To create the boxes, we construct a binary volume from the original dataset. The value of an interior voxel is 1 and all the others are 0. The computation of boxes is done separately for each axis. Since the thicknesses of the boxes are all the same, for an axis, we can cut the volume dataset with a group of uniformly spaced parallel planes perpendicular to the axis. The spacing between adjacent planes is just the thickness of slabs. Then every box is bounded by two of such parallel planes. If an interior voxel v inside a box is known, then the definition of the box will be the smallest axis-aligned bounding box enclosed the set of all possible voxels that can be reached from v without moving through any non-interior voxel or moving out of the two bounding planes or out of the volume dataset.

To simplify the computation, we first project the sub-volume between two parallel planes into an image on either one of the parallel planes. All the voxels lined up parallelly with their facing directions are ORed together to derive the corresponding value in the slice. That is, if any of the voxel along the projection line is 1, the resulting pixel value is 1, too. This can also be done with VolumePro using the maximum intensity projection mode [15]. By replacing the sub-volume with the projection image, we obtain a compressed binary volume, whose size is just $1/d$ of the original volume in the facing direction of the boxes to be generated, where d is the thickness of the slabs.

Next, if a voxel v is known to be an internal voxel, then we can find all reachable voxels with a 2D region-grown algorithm in the slice perpendicular to the facing direction of boxes, using v as the seed. The

[2]In rare cases, the number of the largest faces of a box may be 4 or 6. In either case, the two faces perpendicular to its associated major axis are among the largest.

bounding box of the filled pixels then defines the other four faces of the box. We say the box is seeded from voxel v.

We design an algorithm that traverses all the internal voxels to find all the boxes. The procedure is as follows:

1. pick the first voxel as the current one;
2. It the current voxel is unflagged, find the box seeded from the current voxel, flag all the internal voxels enclosed in the box
3. Set the next voxel as the current, return to step 2.

The above procedure repeats three times with the facing direction of the boxes aligned with all three axes.

Denote the number of total voxels of the dataset is N then the number of voxels in the compressed volume is N/d, where d is the thickness of the slabs or boxes. Assume the number of internal voxels is kN and approximately kN/d respectively. k is number between 0 and 1. Our experience shows that k is approximately 0.17. For the above algorithm, kN/d voxels will be checked whether a region-grown algorithm will be seeded from that voxel. $kN/d + B$ voxels will be checked in the region-grown part and kN/d voxels will be flagged, where B denotes the number voxels outside colon but has at least 1 internal-voxel neighbor, which is definitely smaller than the number of internal voxels. Therefore, the total time complexity of the algorithm is: $6kNT/d$
, where T is the time needed for a read or write operation of a memory cell.

After the boxes are created, for each box, we find the max and min coordinates in the two bordering slices that are perpendicular to its facing direction. Those max and min coordinates are used to determine the front and back portal as shown in 7. For each set of boxes, they are sorted by their values in the facing direction. Consequently, given any given internal voxel v and facing direction, it is trivial to find the box (or boxes, see later discussion) containing v.

3.3.2 Box-assisted Slab Rendering

The rendering process is as following:

1. The system first determines which set of boxes to use depending on which major axis is closest to the current viewing direction.
2. Then for the current camera position (should be inside the colon), find the box containing the camera from the current set of boxes. The box becomes the current box.
3. The sub-volume specified by the current box is clipped by the view frustum and the clipped sub-volume is then the current slab to render.
4. If there is any next box along the viewing direction and it is visible, it is set to be the current box. Go to the previous step; otherwise finish.

   The visibility detection in step 4 is similar to that in [5, 12]. One difference is that in either [5, 12], there is a unique portal on the border of any adjacent boxes. But in our method, when a box B is adjacent with more than one box on the same face, the portal on the shared face of box B encloses the portals on the same face of the adjacent boxes.

Project its near portal onto the image plane. The 2D axis-aligned bounding rectangle is then set to be the current cull window (CCW). Then the boxes visible to the current camera configuration are picked up based on a visibility detection method similar to [5, 12]. Next, each box is clipped by the view frustum to create a slab for the perspective rendering. Each box is associated with two portals, which are used for both visibility detection and image warping as discussed in the previous section. Note in our algorithm, portals of the same box are parallel to each other, which makes the boxes easier to compute. There is such a case in Figure 8, where the portal of box A on plane p is a superset of the portals of B and C on the

same plane. For visibility detection, this causes no problem, since the algorithm computes the intersection of all the portals involved. But the rendering algorithm should take care of multiple next-boxes.

In rare cases, it is possible that one camera position is enclosed in more than one box, as in Figure 9. For such a case, we simply use all the boxes to create slabs, since the opaqueness of colon wall hides the actually invisible sub-volumes.

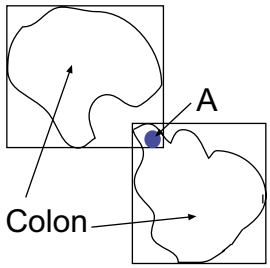

Figure 9: *Overlapped Box.*

3.4 Parallel Processing

In our system there are three major processors involved in the rendering, CPU, VolumePro board and texture mapping hardware. Their workloads are as follows:

· CPU: visibility detection, slab determination and creation.

· VolumePro: loading slabs into volume buffer and render slabs.

· Texture Mapping Hardware: Alpha blending slab images.

Figure 10 shows the parallel pipeline. The data transfer in and out of VolumePro involves more than one processor unit, hence should be synchronized. Both the rendering of VolumePro and the texture mapping on our OpenGL board are no blocking function calls. Therefore, we can overlap the operations of the three processors. In experiments, we found in most cases, the time used by VolumePro is the longest, hence the time needed by CPU and OpenGL can be completely hidden. The parallelization provides us with the convenience of using software created slabs.

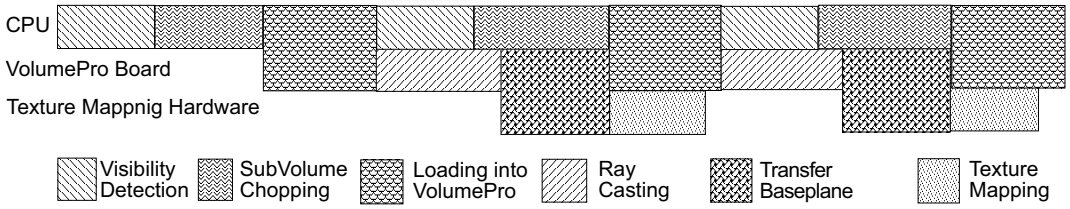

Figure 10: *Parallel pipeline.*

4 Slab Reusing and Preparation

In virtual colonoscopy, the camera is moving along the centerline most of the time, which means the camera parameters are highly correlated in time domain. We can accurately predict future camera parameters, if there is no manual control from the user between the current frame and the frame predicted. This feature as well as the volume alignment of slabs can be used to reuse and prepare slabs. Therefore, we can determine the slabs to render based on the union of the view-frustums of the next one or more images (see Figure 11) if the cost is less than processing every slab separately.

Reusing the slab has two meanings. First, if we use CPU to create the slab and let VolumePro to load it, slabs stored in voxel memory can be rerendered. Second, we can directly reuse slab images if the projection direction is not significantly different.

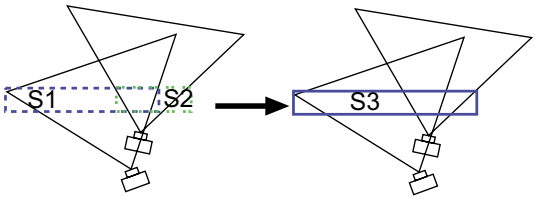

Figure 11: *Slab Merging. Rather than rendering slab S1 and S2 individually, the system renders S3.*

5 Experimental Results and System Performance Analysis

5.1 Performance of VolumePro

To optimize the parameter of our virtual colonoscopy system, such as the size of the boxes, the thickness of slabs, we need to obtain performance parameters of the VolumePro board. The rendering time of VolumePro can be decomposed into four phases. Phase I, loading a volume dataset into the voxel memory; phase II, actual ray casting which should be linear in time with the number of voxels; phase III, overhead for the actual ray casting and phase IV, transfer baseplane image to graphics board for displaying.

Figure 12 shows the rendering time of a series of volumes whose size in the Z dimension, say N, varying from 32 to 256 while keeping constant as 256×256 in the X and the Y dimensions. The "No Display" is obtained without actual display the images by a texture mapping hardware, hence equals the sum of phase II (actual ray casting) and phase III (overhead). Obviously, it is a perfect line. "Single Buffer" and "Double Buffer" are rendering time including display. "Single Buffer" means each time VolumePro renders to the same baseplane. "Double Buffer" means that VolumePro alternatively renders into two baseplanes. Therefore, while VolumePro creating an image on one baseplane, The texture mapping hardware can read from the other baseplane. We consider the difference between "Double Buffer" and "No Display" as the time required for transferring a baseplane image to OpenGL. The data of $size = 0$ are extrapolated.

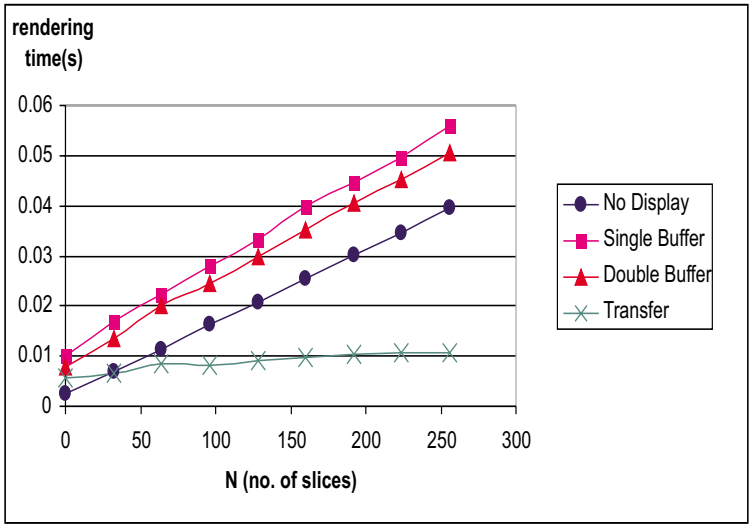

Figure 12: *Ray Casting Performance of* $256 \times 256 \times N$ *Volume by VolumePro.*

Figures 13a and b displays the time to render 32 byte thick slabs without loading the volume and displaying rendered baseplane image. As expected, the time is linear in both X and Y, as:

$T(X, Y) = k_1XY + k_2X + k_3Y + k_4$

Where, k_1, k_2, k_3, k_4 are constants. By solving a linear equation system, we obtained: $k_1 = 1.01 \times 10^{-7}, k_2 = k_3 = 1.13 \times 10^{-6}, k_4 = 2.59 \times 10^{-3}$.

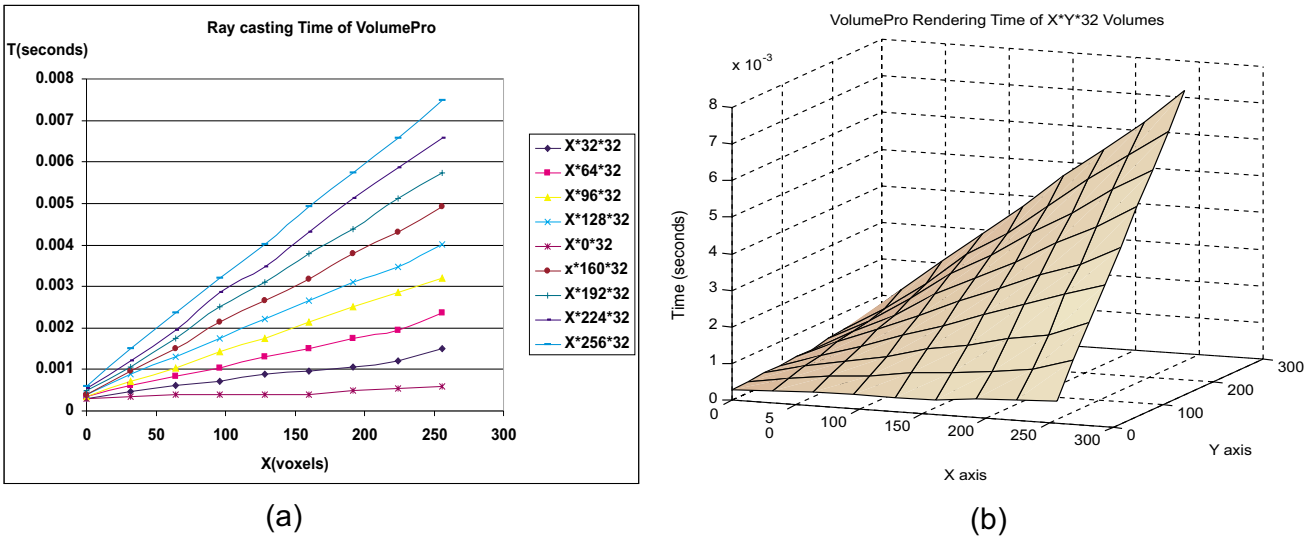

Figure 13: *Ray Casting Performance of* $X \times Y \times 32$ *Volume by VolumePro.*

Figure 14 is the statistics of creating and loading subvolumes. The difference between the two curves is the time for transfering data from host memory to voxel memory, which involves both CPU and VolumePro.

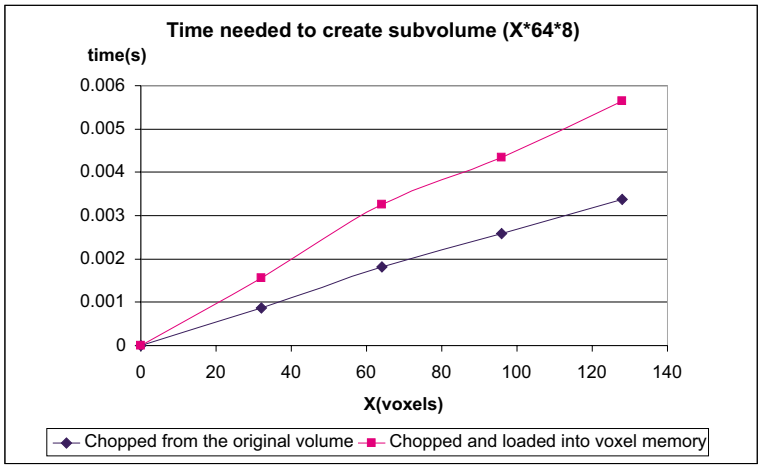

Figure 14: *Time of creating* $X \times 64 \times 8$ *volumes.*

5.2 Performances of the Virtual Colonoscopy System

The frame rate of the virtual colonoscopy system varies significantly according to several parameters. When the camera is looking at one end of a long tube and looking towards the other end, the system need to render more slabs; while the camera is look at the colon wall, only a few slabs need to be handled. One good thing is that larger number of slabs occurred only when the navigation direction is parallel to the centerline, it is the time more slabs can be reused.

Since the ActiveSubVolume function, which requires 32 bytes alignment, is still a choice in the future and the thinnest slab in the previous system is 6 voxels, we choose each slabs to be 8 voxel thick, thus

one dimension of the slabs are constant. We also observed that the radius of colon varies from 1 to 20 voxels, therefore, we are sure there is at least another dimension that is equal to or less than 64 voxels, no matter of the colon tube orientation. Even for the highest speed of flying, the camera never proceeds more than 8 voxels from one frame to another, therefore, the system loads at most one new slab for each new frame except that the viewing direction changes significantly and the system switches to a different set of boxes. The switching happens usually in two cases, (1) user turns purposely the viewing direction and makes the camera facing the colon wall; (2) the centerline turns abruptly and the navigation enters a long tube. For the first case, the viewable depth is generally much less than viewing along the centerline since the diameter of colon is less than 40 voxels, therefore no more than 8 slabs are needed to be rendered. For the second case, the switching of boxes only applies to the first frame of entering the tube, and there is only a few long pipes in a colon dataset, typically less than 10. That is to say, only a small percentage of frames during the entire navigation exhibit significant delay (typically less than 1%).

The longest line segment that can be put into the colon is less than 256 voxels and the average of such length is approximately 160. Consequently, we can evaluate the frame rate by imagining the rendering is inside of a $160 \times 64 \times 64$ or smaller tube. The number of slabs needed to render such a tube depends on the viewing position and direction as well as the direction of the centerline. Since the number of voxels for a tube is fixed, the more slabs, the slower the rendering. First, let's consider flying along the centerline. The system may exhibits delays for the first frame since VolumePro may have to load and render all the slabs and when the camera flies into a long pipe. The average of such delay is 0.079 second, the equivalent of 12 frame/s. Fortunately, for the rest of frames inside the pipe, the frame rate is much higher. The total number of slab images is $160/8 = 20$. Assume system loads one new slab of the size of $128 \times 64 \times 8$ (usually, the newly loaded slab size for flying along the centerline is $64 \times 64 \times 8$) and rerender the new slab as well as the 10 slabs closest to the camera owing to the change of viewing direction. For the hardware configuration listed above, the frame rate is: $38Hz$. Moreover, the rerender of existing slabs is usually unnecessary because the flying direction stays in a small range. Therefore, the actual rendering speed may be as big as 50 frames per second. When the camera is facing the colon wall, assume the system reloads and render each slab, the rendering speed is still greater than 23 frames per second.

The navigation in a real colon dataset verifies our analysis by showing a frame rate of over 30 frames per second and usually varying in the range of 20 to 40 frames/s.

6 Discussion and Future Work

In this paper, we present a virtual colonoscopy system based on VolumePro, a real-time volume-rendering PC board. By proposing image-warping assisted perspective rendering, slab oriented boxing, a software controlled pipeline as well as slab reusing and prediction, we can navigate inside the colon in real-time with image quality comparable to what we have achieved on an expensive SGI super-computer, whereas on a 16-cpu SGI challenge (9 processors exploited), the frame rate is only about 2 frames/s.

The design of slabs and boxes makes it natural to extend the algorithm to an image-based rendering scheme, since the last stage of the rendering pipeline is just texture mapping and warping slab images. We can prerender and store the images of all the boxes for all the possible viewing angles, then use texture-mapping hardware to composite appropriate slab images. Thus, we can build a virtual colonoscopy system without a VolumePro board. The rendering of boxes can be done with any volume rendering algorithm. We can design a client-server architecture, with a powerful server (perhaps equipped with VolumePro) to prerender box images and several ordinary computers to show the navigation.

7 Acknowledgments

This work has been supported by grants from NIH #CA79180, Office of Naval Research under grant N000149710402, and E-Z-EM Inc. The pipe and patients' data sets were provided by the University Hospital of the State University of New York at Stony Brook.

References

[1] J. Mandel, J. Bond, J. Church, and D. Snover (1993). "Reducing Mortality from Colon Cancer Control Study". *New England J Med 328*, 1993, 1365-1371.

[2] W. Lorensen, F. Jolesz, and R. Kikinis (1995). "The Exploration of Cross-Sectional Data with a Virtual Endoscope". In R. Satava and K. Morgan (eds.), *Interactive Technology and the New Medical Paradigm for Health Care*, IOS Press, Washington D. C., 1995, 221-230.

[3] D. Vining, D. Gelfand, R. Bechtold, E. Scharling, E. Grishaw, and R. Shifrin (1994). "Technical Feasibility of Colon Imaging with Helical CT and Virtual Reality". Presented at *the Annual Meeting of the American Roentgen Ray Society*, New Orleans, April, 1994.

[4] L. Hong, A. Kaufman, Y. Wei, A. Viswambharn, M. Wax, and Z. Liang (1995). "3D Virtual Colonoscopy". *Proc. 1995 Symposium on Biomedical Visualization*, 1995, 26-32.

[5] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He (1997). "Virtual Voyage: Interactive Navigation in the Human Colon". *Proc. SIGGRAPH '97*, August 1997, 27-34.

[6] S. You, L. Hong, M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang (1997). "Interactive Volume Rendering for Virtual Colonoscopy". *Proc. IEEE Visualization '97*, Phoenix, AZ, October 1997, 433-436.

[7] M. Wan, Q. Tang, A. Kaufman, Z. Liang, and M. Wax (1999). "Volume Rendering Based Interactive Navigation within the Human Colon". *Proc. IEEE Visualization' 99*, 1999, 397-400.

[8] M. Wan, W. Li, K. Kreeger, I. Bitter, A. Kaufman, Z. Liang, D. Chen and M. Wax. "3D Virtual Colonoscopy with Real-time Volume Rendering". To appear on *Proc. SPIE Medical Imaging 2000.*

[9] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami (1997). EM-Cube: An Architecture for Low-Cost Real-Time Volume Rendering. *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 131–138, ACM Press, August 1997.

[10] H. Pfister, J. hardenbergh, J. Knittel, H. Lauer, L. Seiler (1999). "The VolumePro Real-Time Ray-Casting System". *Proc. SIGGRAPH '99*, August 1999, 251-260.

[11] H. Pfister and A. Kaufman (1996). "Cube-4: A scalable architecture for real-time volume rendering". *Proc. IEEE Symposium on Volume Visualization' 96*, 47-54, ACM Press, October 1996.

[12] A. Kaufman, F. Dachille IX, B. Chen, I. Bitter, K.A. Kreeger, N. Zhang, Q. Tang and H. Hua, "Real-time volume rendering", to appear in the *International Journal of Imaging Systems and Technology*, special issue on 3d Imaging, 1999.

[13] K.A. Kreeger, I. Bitter, F. Dachille, B. Chen, A. Kaufman. "Adaptive Perspective Ray Casting". *Symposium on Volume Visualization.* Research Triangle Park, NC. October 19-20, 1998.

[14] K. Mueller, N. Shareef, j. Huang and R. Crawfis. "IBR-Assisted Volume Rendering". *Late Breaking Hot topics session at Visualization'99.*

[15] RealTime Visualization, Inc. "Volume Library Interface User's Guide, A Guide to Programming with VolumePro v. 1.0".