Visual Comput (2007) 23: 545–557
DOI 10.1007/s00371-007-0117-8

ORIGINAL ARTICLE

Seok-Jae Jeong
Arie E. Kaufman

Interactive wireless virtual colonoscopy

Published online: 23 May 2007

S.-J. Jeong (✉) · A.E. Kaufman
Center of Visual Computing, Dept. of Computer Science, State University of New York at Stony Brook, USA
seokjae@gmail.com, ari@cs.sunysb.edu

Abstract We present an interactive virtual colon navigation system on a PDA that is a client-server system over a wireless network. For improving the quality of the rendering results on the PDA, the overall rendering speed, and the user interactivity, we propose three novel methods and adapt a GPU-based direct volume rendering technique. Using these proposed methods, our system can support approximately a two times faster navigation speed and 17 percent better PSNR than previous remote visualization methods with a $512 \times 512 \times 361$ volumetric colon CT data using a PDA device over 802.11b wireless network.

Keywords Wireless virtual colonoscopy · Remote visualization · GPU-based volume rendering · Interactive image streaming

1 Introduction

During the last decade, numerous publications related to 3D virtual colonoscopy have been published in the fields of medical science and computer graphics [2, 9, 12, 14, 22, 28, 29]. 3D virtual colonoscopy enables radiologists to navigate virtually inside a patient's colon in real-time using a contemporary PC. Virtual colonoscopy (VC) is a computer-graphics-based application developed as an alternative to conventional optical colonoscopy [12, 14]. Typically, VC takes 350–750 abdominal axial CT images of 512×512 sub-millimeter resolution using a helical or multi-slice CT scanner. From the CT scan, a 3D model of the colon is constructed by extracting the colon out of the rest of the abdomen with automatic segmentation and by removing the residual material inside the colon with an electronic cleansing algorithm. The user, typically a physician or a radiologist, can navigate virtually inside the constructed 3D model of the colon in real-time using a PC-based visualization software based on volume rendering. In contrast to optical colonoscopy, VC is patient-friendly and provides a fast, non-invasive, accurate, and cost-effective method for detecting colon polyps, the precursors of colon cancer. We have adapted the direct volume rendering method to render the colon endoscopic image in order to improve the visualization and the detection of polyps [28].

In this paper, VC has been extended to a wireless environment with a mobile device, such as the PDA, shown in Fig. 1. This allows the user to interactively navigate inside the colon through a wireless network away from the PC workstation. Wireless virtual colonoscopy (WVC) follows exactly the same procedure as VC for pre-processing and constructing a 3D model of the colon. However, WVC requires additional functionality for the remote navigation inside the colon. First, it should provide not only the rendering of an endoscopic view but also transmit the rendering results to the mobile device, since the mobile device usually does not have enough processing power to gener-

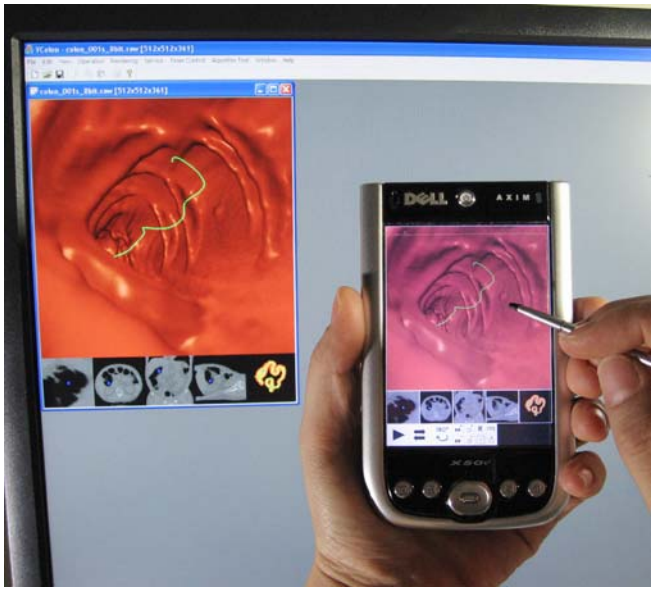

Fig. 1. Our interactive wireless virtual colonoscopy running on the server (in the *background*) and the client (in the *foreground*)

ate a volume rendering image in real-time. Therefore, the generation of endoscopic view images needs to be accelerated on the server side as these images also need to be transmitted. One must further consider the time necessary for displaying transmitted images on the client mobile device. Second, WVC should provide a novel user interface to display interactive navigation results on the mobile device on a low resolution screen, 480×640 or lower.

There have been many attempts to expand 3D graphics applications to remote visualization systems for displaying results on thin-client computers. Ang et al. [1] extended the capabilities of the web to include access to 3D volume data sets with integrated control of a visualization cluster system. Their system provides a web-based user interface and utilizes a web browser to visualize volume data. They also mentioned that for future work MPEG may be used to stream the result images. Lippert et al. [18] proposed that the server compresses the volume data based on wavelet splats. The client then carries out all rendering directly in the compression domain. Engels et al. [6] presented a framework for web-based visualization using image streaming techniques, which enables the remote control of an OpenInventer application. In their framework, the client sends visualization parameters and GUI events to the server and the server transfers compressed rendering images to the client. They extended their framework to a Cosmo3D-based visualization application and applied several compression techniques to transmit the rendered results [8]. For real-time volume rendering, they used the 3D texture-based volume rendering method and this framework has been tested over a wireless network as well as a local area network [7].

There also have been many reports on the client-server-based visualization of 3D scenes using PDA over a wireless network. Lamberti et al. [16] proposed a generic solution for hardware-accelerated remote rendering on the visualization cluster. To accelerate server side rendering, they used the chromium architecture as a PC cluster and adapted ZLIB and JPEG compression algorithms to reduce the transmission data size. Lamberti and Sanna [17] expanded their previous work by adapting the MPEG transport stream (MPEG-TS) standard for streaming rendered images with UDP protocol.

The client-server based visualization solutions render a 3D scene on a server and transmit the rendered result to the thin client device over a network. Thus, the performance of these image-based approaches strongly depends on the resolution of the rendering result. The above-mentioned methods adapted several compression algorithms to compress the resulting images in order to improve the transmission performance. However, adapting compression algorithms requires additional costs to both the client and the server. In addition, it degrades the final image quality on the client when a lossy compression algorithm is used.

Diepstraten et al. [5] presented an alternative method to the image-based approach by splitting the workload between client and server and reducing the required network bandwidth for every frame. In their system, the server extracts feature lines from the rendered image and sends it to a PDA client with vector information. They obtained very high frame rates using the GAPI library rather than the GDI library on the PDA. Quillet et al. [23] adapted a similar idea to visualize large city models on the PDA client. However, they used a level of detail technique to reduce the number of rendering lines depending on the viewing position. However, the above two approaches enable the user to see only a non-photo realistic image with 2D lines on the client.

Recently, substantial research has been conducted on real-time direct volume rendering using commodity graphics cards. Krüger and Westermann [15] proposed an accelerated volume ray-casting technique using the graphical processing unit (GPU). They exploited the 3D texture, early z-test, and fragment shader to implement a GPU-based ray-caster. Their acceleration technique works on multi-pass rendering with Shader Model 2.0. Stegmaier et al. [27] presented a flexible framework for GPU-based volume rendering. In their framework, DirectX Shader Model 3.0 was used for single-pass volume ray-casting.

Our interactive wireless virtual colonoscopy provides the doctor with more flexible access to the virtual colonoscopy than previous stand-alone virtual colonoscopy systems. In other words, the doctor can virtually navigate a patient's colon, interactively looking for polyps anywhere and anytime on a mobile device over a wireless network. Our WVC was able to support multi-user accessibility with a high performance GPU and some additional

implementation. Thus, it enables users to share valuable resources on the server. Eventually, it may be a useful application in ubiquitous computing environments.

In this paper, we propose the following three novel methods to improve the quality of the rendering result on the PDA, the overall rendering speed, and the user interactivity of the WVC.

- *Separating transmission of image and vector data*: The server sends the rendering results with image data and vector data separately, which makes the client display a higher quality result than previous methods, especially when lossy image compression methods are used.
- *Requesting a series of frames*: To achieve higher frame rates for displaying the rendering results on the client side, the client sends a request for a series of rendering results while navigating a colon along the colon centerline. Using this scheme, the transmission time can be reduced by sending multiple frames at once from the server to the client without loss of interactivity.
- *Generating intermediate frames*: For pointing pen dragging operations such as the rotation of a viewing direction or in/out zoom, the server generates intermediate frames automatically between successive positions of the drag. With this method, the user is provided with smoother interaction than with previous methods.

In addition, for the real-time rendering on the server-side, we utilized a GPU-based volume rendering method using the fragment shader with a 3D distance field for generating the endoscopic image.

The remainder of this paper is organized as follows. In the next section, we describe the implemented client–server system architecture and its user interface. Then, we explain the proposed methods in Sects. 3 to 6. The performance test is presented in Sect. 7. Finally, we conclude this paper with directions for future work in Sect. 8.

2 Interactive wireless virtual colonoscopy

Figure 2 shows the implemented wireless virtual colonoscopy system architecture. The client PDA connects and sends requests to the server through a wireless access point. To accomplish real-time volume rendering on the server, the server is equipped with the recent commodity PC graphics hardware supporting Shader Model 3.0.

The service procedure is as follows. The client first sends a request to the server and the server loads the colon data consisting of a 3D CT volume, a 3D distance field volume, a colon surface mesh and a centerline. These data are transferred to the GPU memory; the CT volume and the distance field volume data are converted to 3D textures

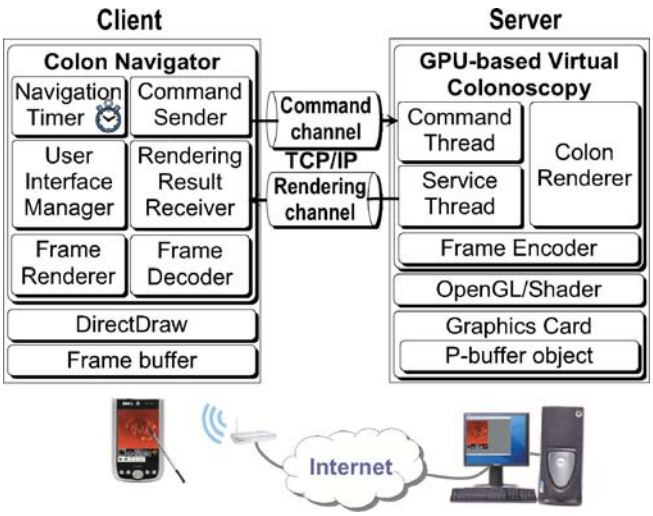

Fig. 2. Interactive wireless virtual colonoscopy system architecture

and the mesh and the centerline are display lists. After loading, when the data is ready to navigate the colon the server sends an ACK to the client with a *client-ID*. The *client-ID* is used so that the server can identify the client. Through the above stages, a communication channel is constructed for transferring rendering results. We call it the "rendering channel", which is managed by one thread in the server called the "service thread".

The client sends another request with the client-ID to construct another communication channel; we call it the "command channel", which is used for transferring commands from the client to the server. On the server, commands transferred from the client are queued and processed sequentially by another thread, which is called the "command thread". The server has a timer to trigger the service thread for processing queued commands periodically. After constructing both channels, the client sends a first rendering request to the server through the command channel.

When the user starts the navigation, the client sets up the *navigation timer* to trigger a navigation request, which enables the program to process user's input during the navigation without a thread implementation. The *user interface manager* converts an input from the user into a request and the *command sender* transfers it to the server. The *rendering result receiver* receives the encoded frames from the server and conveys them to the *frame decoder*. Finally, the *frame renderer* displays the decoded frames on the *frame buffer* using *DirectDraw*.

For every rendering request, the server first performs the GPU-based direct volume rendering to generate an endoscopic image during the navigation. A fixed number of camera positions are uniformly placed on the centerline and the navigation is progressed by following the camera positions. Whenever an endoscopic image is generated, the CPU computes the viewing direction of the

current camera so that the direction points from the current camera position to a constant- distanced forward camera position. Using the computed viewing direction, the CPU computes the information of the image plane from which rays will be cast. The image plane information is passed to the fragment program with two 3D textures and one 1D texture, which includes transfer functions for the direct volume rendering algorithm.

After rendering the endoscopic image, the server renders four slice images (axial, coronal, sagittal, cross sectional) by rendering the four axial, coronal, sagittal, and cross sectional quadrangles with the 3D CT volume texture in OpenGL. Finally, it renders a colon surface overview image with the colon surface mesh and the centerline. As is shown in Fig. 3a, the server sends the six images, and the client receives them through the rendering channel and displays them. The user can see the received images with the user controls shown as Figs. 3b–3g.

The system has the following user interface. The user can start and stop the navigation by tapping the play/pause button, which is the leftmost button in the user control panel, shown in Fig. 3b. In Fig. 3b, the area of the displayed endoscopic image is called the "main view". The other areas displaying small images are called "small views". The user can change the image to appear in the main view by touching the small view with a pointing pen. Figures 3b–3g shows six different types of images of the virtual colonoscopy in the main view. For the endoscopic view and the colon overview image, interactions such as rotating the view direction or zooming in/out occurs when the user drags a pointing pen in the main view. The navigational direction can be changed to move backwards from forwards and vice versa using the changing navigation direction button, which is the second leftmost button in the user control panel. In addition, the user can change the viewing direction by 180 degrees using the change viewing direction button, which is the third leftmost button in the user control panel.

The system provides two kinds of frame size: the *full frame size* and the *half frame size*. The full frame size, 480×576, is a result of the PDA VGA resolution of 480×640. In the full frame size, the resolution of the main view is 480×480 and small views have 96×96 resolution each. Similarly, the half frame size, 240×288, results from QVGA resolution of 240×320. In the half frame size, the resolution of the main view is 240×240 and the small views are 48×48 each. The size of the frame can be selected by the user at the client or the server.

There is another frame size mode, which is the automatic mode. If the user sets the frame size to this mode, the system transfers the rendering frames in the half size for the moving cases but in the full size for the paused cases, such as the final frame of the rotation or zoom in/out request and the response frame for the pause navigation request. The automatic mode improves the overall rendering speed and thus gives the user better interaction.

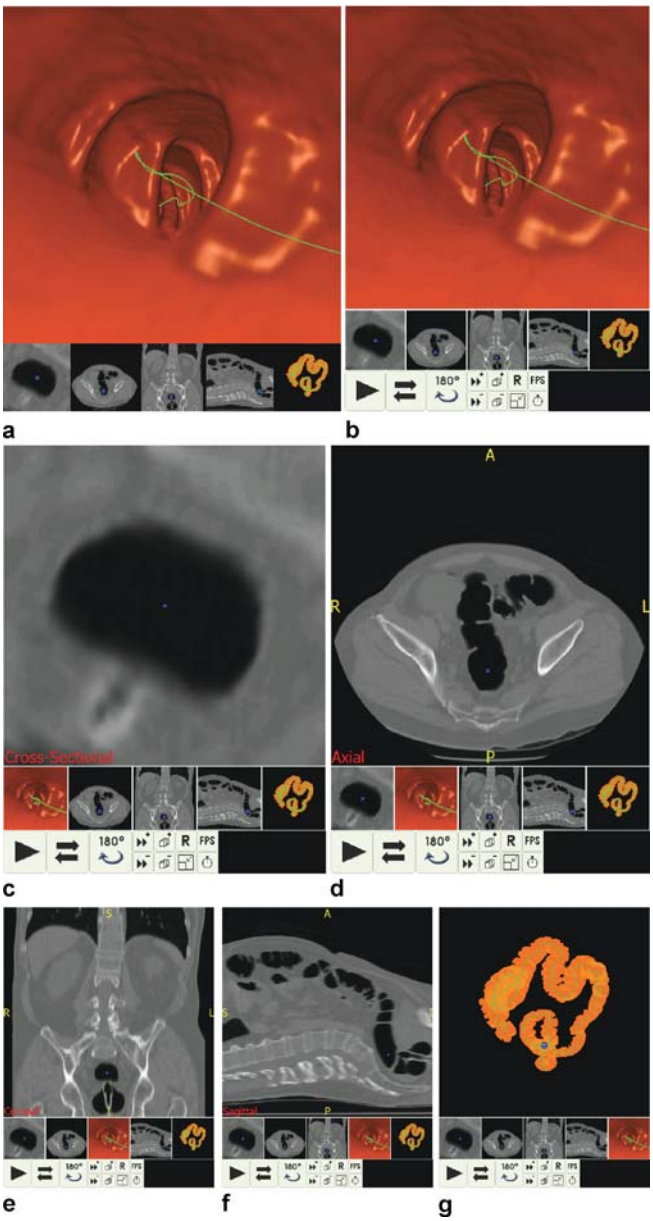

Fig. 3. a The server renders six types of images and organizes them into one image frame depending on the current main view image type; **b–g** the client displays the received image frame with user controls; the user can choose the type of image to appear in the main view by tapping one of the corresponding small views; endoscopic image (**b**), cross sectional slice image (**c**), axial slice image (**d**), coronal slice image (**e**), sagittal slice image (**f**), and surface overview image (**g**)

3 Separating transmission of image and vector data

As mentioned in Sect. 1, most of the client–server based visualization solutions generate the rendering results as 2D images on a server and transmit them to the thin client

device. As a result, the client simply displays the transferred images. Here, we consider the processing power of the client device for the rendering. In other words, the server can pass part of rendering burden to the client, if possible. However, the direct volume rendering methods used in our application treats very large data and requires a very high processing power. Therefore, it is difficult to share the 3D model through the low bandwidth network environment such as wireless LAN and to utilize the client resource for the volume rendering.

Even though Diepstraten et al. [5] presented a method of splitting the workload between the client and the server, they only transferred the extracted lines from a rendered image. Thus, a non-photorealistic image is presented to the user. We propose a novel approach to utilize the client resource for showing volume rendering results combined with drawn vector data to the user. The basic idea is rather simple. First, the server generates a rendering result as a 2D image and its vector data, which can be drawn rapidly on the client, such as lines and points. The rendered 2D image is then compressed by an image compression method, such as JPEG. Second, it sends the compressed 2D image and the vector data separately to the client. Finally, the client decompresses the compressed 2D image, displays it and draws the vector data on top of the image.

In our application domain, the centerlines in the endoscopic image and the colon surface overview image are vector data. In addition, the current camera position in the slice images is further vector data. For example, the server first generates a rendering result as a 2D image, shown in Fig. 4a, and compresses it with a JPEG compression method. Then, it generates simple vector data enough to rapidly render on the PDA; this is shown in Fig. 4b. The client receives the vector data as well as the 2D image data and displays them together on the screen. Figure 4c shows a rendering result on the PDA.

Using this method, we can obtain high quality in rendering results even with a lossy image compression algorithm, such as JPEG. For the case of JPEG, high frequency areas such as the boundary between the colon lumen surface and the centerline in the colonoscopy image are usually compressed with artifacts, because the JPEG compression quantizes DCT coefficients.

Figure 5a shows the transferred rendering result by our method and Fig. 5d the rendering result by JPEG compression only. Figures 5b and 5e show the enlarged images around the centerline in the endoscopic view image by our method and the image data only method, respectively. In the same way, Figs. 5c and 5f show the enlarged colon surface overview images by our method and the image data only method, respectively. Our method demonstrates better quality around the centerline than the image data only method. Both of the images compared in Fig. 5 are quant-

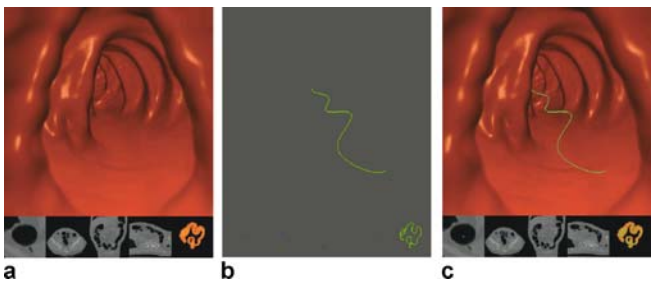

Fig. 4a–c. Separating image data (**a**) and vector data (**b**) and displaying them together on the client (**c**)

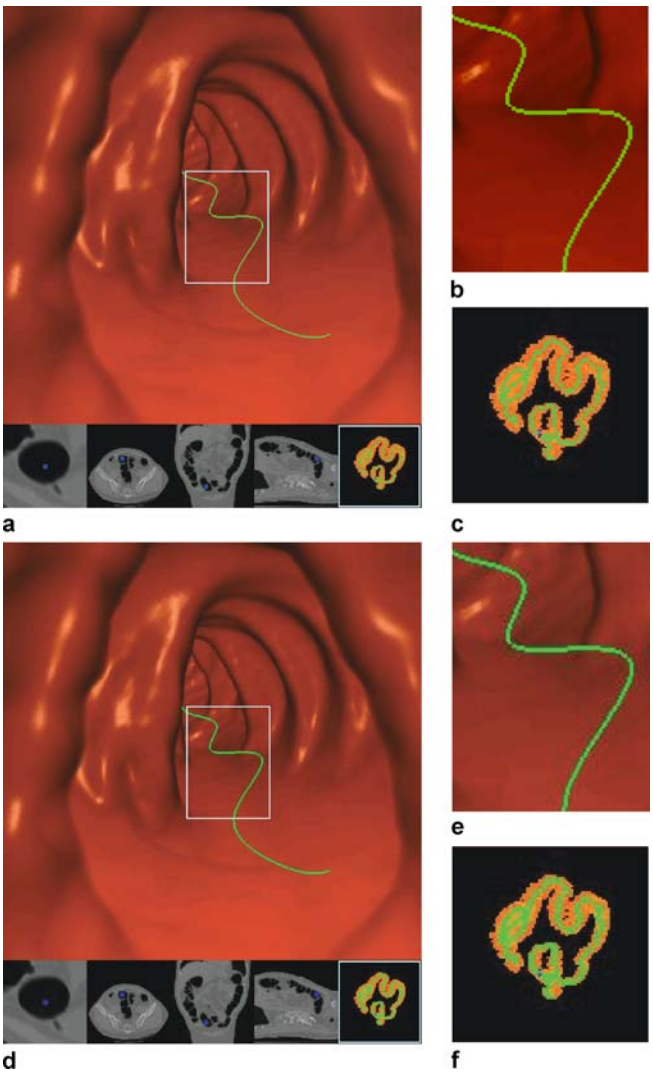

Fig. 5. **a** Transferred rendering result using separate vector and image data; the enlarged image around the centerline (**b**) and the enlarged colon surface overview image (**c**) from **a**; **d** transferred rendering result by sending image data only; the enlarged image around the centerline (**e**) and the enlarged colon surface overview image (**f**) from **b**

ized to 16 bits per pixel because the PDA currently only supports up to 16-bit colors.

4 Requesting a series of frames

In previously proposed remote visualization systems, the client usually sends a request for each rendering step. However, the client does not have to send requests for all rendering steps in the case of navigation following a known predefined path. Instead, the client notifies the server to start the navigation by sending a simple request and the server can send rendering results continuously until it receives a notification from the client to stop the navigation. However, such a continuous transmission scheme from the server to the client causes network congestion, especially when it takes much longer time for the client to display a transferred frame than for the server to generate it.

PDA has much less processing power than a PC workstation. Furthermore, the transferred frames are usually compressed to reduce the data size and decompressing them consumes a fairly long time on the PDA. We were able to solve this problem by briefly delaying the server between consecutive transmissions. However, it is difficult to determine the exact time for the delay in the wireless environment because the delaying time should be changed whenever congestion takes place, which could occur at any time depending on the network environment.

We propose an alternative approach to accelerate the transmission speed from the server to the client. In the proposed method, the client sends a request for a series of consecutive frames. Then, the server responds to the client with a set of rendering results. This method is very efficient for transfer and display in the case of navigation following a predefined path. However, we have to solve two problems for adapting this method for interactive navigation. First, the number of consecutive frames to be transferred should be determined. Second, when the user commands a pause in navigation in the middle of displaying the consecutive frames, the navigation should be stopped. Moreover, if there is another command given by the user after the pause command, the server has to generate the next rendering frame based on the paused frame.

We have determined the number of consecutive frames from experimental results, which will be described in Sect. 7. In addition, to enable the user to pause the navigation in the middle of displaying transferred consecutive frames, the server stores camera positions and the information of the image planes of the generated series of frames. This information is used for rollback to the rendering phase of the paused frame. When the user stops the navigation while the received frames are being displayed, the client stops displaying, ignores the remaining undisplayed frames yet and sends a pause-request to the server with the index of the paused frame. The server can then rollback using the stored information. Thus, the next rendering is performed from the rollback position. Figure 6 shows the procedure for this scenario.

5 Generating intermediate frames

Whenever a user drags a pointing pen in the main view to rotate the viewing direction or to zoom in/out, the client catches drag events that occurred during the action and sends a rendering request for every event to

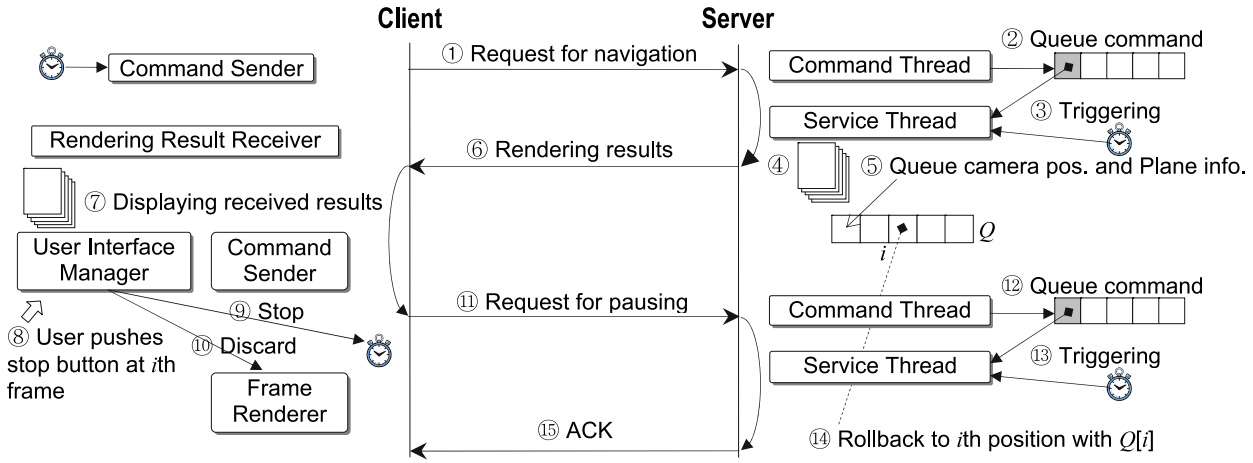

Fig. 6. Scenario of pausing the navigation with our proposed method; when the user stops the navigation while the received frames are being displayed, the server rollbacks using the stored camera positions and image planes information of the series of consecutive frames. Then, the server renders the following frames from the rollback position

the server. A previously proposed system [16] generated a rendering result per request. However, this method yields the temporal aliased sequence of frames because the drag events occur more sparsely while the client program is busy receiving and displaying the rendered frames.

We propose an alternative approach to improve the interactivity. In our method, the previous pen position and the current one are sent to the server from the client as the start position and the end position, respectively. The server then computes several evenly spaced positions between the two positions, generating rendering frames for these positions and the ending position, and transferring them to the client. This method provides smoother interactivity for the user than the previous method because it provides the user with intermediate frames between the pen moving events, as well as a frame at the end position. In addition, it does not cause serious overhead because the client sends one request to the server for each set of consecutive frames, as discussed in Sect. 4. The number of generated positions (N) is computed as follows:

$$N = \max(1, \lfloor \log_2(\max(|\mathrm{d}x|, |\mathrm{d}y|)) \rfloor),$$
$$\mathrm{d}x = x_e - x_s, \quad \mathrm{d}y = y_e - y_s, \tag{1}$$

where the start position is (x_s, y_s) and the end position is (x_e, y_e).

Consequently, the generated positions (P) are computed as follows:

$$P_i = \left(x_s + \mathrm{d}_x \times \frac{i}{N}, y_s + \mathrm{d}_y \times \frac{i}{N} \right), \quad i = 1, \ldots, N. \tag{2}$$

Figure 7 shows an example of the generated positions when a user drags a pen in the main view for rotating the viewing direction. Let us assume that four events occurred at the positions of the squares in Fig. 7 during the user pen drag from the lower left corner to upper right corner along the solid path. These four positions are (85, 415), (110, 280), (247, 130), and (383, 115). The client sends a request for rotating the viewing direction with these positions. Then, the server computes the positions of the circles between the squared positions shown in Fig. 7. Consequently, the server generates rendering results for all squared and circled positions except the starting squared position.

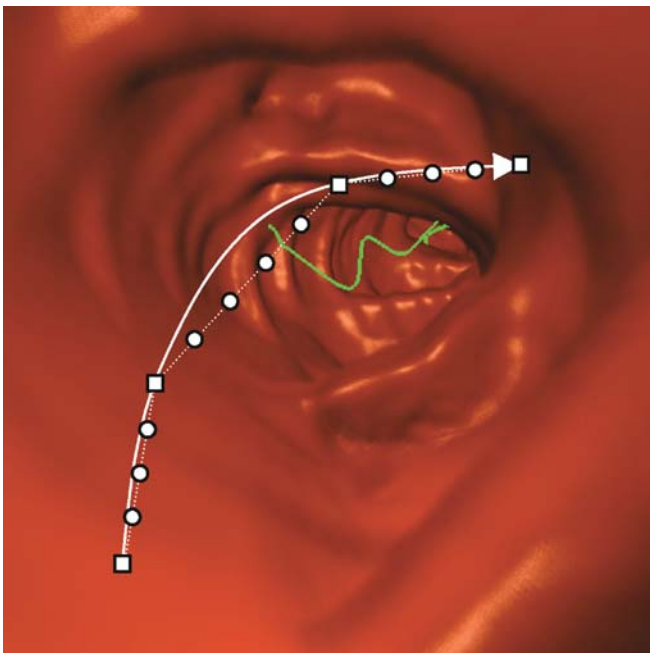

Fig. 7. An example of the positions generated by a user's pen-dragging action; the user drags along the solid arrow line and the client generates four events on the positions of the squares (□). The server then automatically computes the intermediate positions marked as circles (○) along the dotted line and generates rendering results for all squared and circled positions except the starting squared position

6 Ray-casting with a 3D distance field

Ray-casting is a time-consuming process for volume rendering. For this reason, there have been many acceleration techniques for skipping the empty space inside and outside of the volume. Krüger and Westermann [15] exploited the volume bounding box to compute ray directions and determine the entry and exit points of the rays. In addition, they used 1/8 of the original volume to encode empty regions. For the virtual colonoscopy application, Wan et al. [28] used the distance field for skipping empty space inside the colon.

We used the 3D distance field to accelerate our GPU-based ray-casting algorithm in the fragment shader of a 480×480 endoscopic image. To do this, the server loads not only a 3D colon CT volume but also another volume for the 3D distance field. In the 3D distance volume, each voxel holds the distance to the closest colon surface voxel if it is inside of the colon. Otherwise, it has the value of 0. In our implementation, we used one byte per voxel. Thus, the maximum distance value is 255.

The 3D distance field is computed by the method proposed in [19]. It takes about 106 seconds to compute the distance field of a $512 \times 512 \times 361$ volume data off line. The server loads it to the GPU along with the colon data. Finally, it is passed to the fragment shader for ray-casting. To compute the initial ray position and ray direction in the fragment shader, the CPU computes the image plane information where each ray will be cast, as well as the camera position, and passes them to the shader. The procedure of the fragment program including our proposed acceleration technique is as follows.

1. Compute the current ray position, `vPos`, with the image plane information;
2. Compute the current ray direction:
   `vRayDir = vPos - vCamera;`
3. `while (fOut.color.a < 0.99) {`
   `// Get the value for empty space skipping;`
   `skip = tex3D(texVolDist,vPos).a;`
   `if (skip > 0) {`
   `vRay += skip*255;`
   `dist += skip*255;`
   `}`
   `else {`
   `inten=tex3D(texVol,vPos).a;`
   `rgba=tex1D(texTrans,inten);`
   Compute the normal vector;
   Compute shading color and opacity;
   Accumulate shading color and opacity into `fOut.color`;
   `vPos += fRayStep * vRayDir;`
   `dist += fRayStep;`
   `}`
   `}`
4. Output the accumulated color and the distance value.

The `skip` value is multiplied by 255 because the texture values are normalized within 0 to 1. The variables `texVolDist` and `texVol` are the distance field volume and the CT volume, respectively. The transfer function is `texTrans`, which is a 1D texture. The variable `dist` is used for accumulating the distance from the image plane to the voxel last intersected by the ray. The accumulated color values are stored in the frame buffer and the distance values are stored in the depth buffer. However, the depth value must be adjusted to the depth in a perspective projection. The final depth value (z) is computed by Eq. 3.

$$z = \left(\frac{1}{z_{\text{near}}} - \frac{1}{\texttt{dist}}\right) \Big/ \left(\frac{1}{z_{\text{near}}} - \frac{1}{z_{\text{far}}}\right). \tag{3}$$

The z_{near} and z_{far} values are the distances between the camera position and the clipping planes consisting of the perspective viewing volume. The z value is used for the depth test to overlay the centerline on the endoscopic image considering the depth. In Sect. 7, we will show the comparison of performance results with and without the 3D distance field for the GPU-based raycasting.

7 Experimental results

In this section, we show four experimental results related to the four proposed methods, respectively. We first implemented a wireless virtual colon navigation system based on [16] to compare it with our methods. The tests have been conducted under the following system environment. First, the server has dual Intel 3.60 GHz Xeon CPUs, 3 GB RAM, and an NVIDIA Quadro FX 4500 graphics card. It is operated on Microsoft Windows XP Pro. Second, the client, Dell Axim X50v, has an Intel PXA270 CPU 624 MHz, 64 MB RAM, and an Intel 2700G multimedia accelerator. The operating system of the client is Microsoft Windows Mobile Version 5.0. Finally, the communication between the client and the server occurs via a 802.11b wireless LAN. The client and the server programs were implemented with C/C++. For the real-time full volume rendering at the server side, the rendering procedure was implemented with OpenGL with Shader Model 3.0 [21]. The client used Microsoft DirectDraw and Microsoft Imaging API for displaying the transferred frames [20].

To evaluate previously proposed methods, we adapted the following compression methods for our system.

- *RAW*: This is not a compression method. It is implemented as a base line to compare with the compression algorithms.
- *ZLIB*: Lossless data compression based on a deflate algorithm. The zlib data format is itself portable across platforms [11].
- *PNG*: Portable network graphics is an open, extensible image format with lossless compression. It also supports compression for 8-bit color images as well as 24-bit color images [25].
- *JPEG*: This is the most popular image compression standard. Usually, JPEG is used for lossy compression of images [13].
- *FS Dithering*: Floyd–Steinberg dithering is used for quantizing 24-bit color images to 8-bit color images. This algorithm achieves dithering by diffusing the quantization error of a pixel to its neighboring pixels [10].
- *FS Dithering and PNG*: Compression with PNG after FS dithering.
- *CCC*: Color cell compression yields reasonable color images in approximately two bits per pixel from 24-bit color images. Furthermore, the simplicity of this algorithm enables the PDA to decode images fast [4].
- *CCC and ZLIB*: Compression with ZLIB after CCC.

For all types of compression methods, the final rendering results are quantized to 16 bits per pixel images due to the limitation of the PDA device.

Our tests were conducted with a $512 \times 512 \times 361$ CT volume data, which has 90.25 MB. The same size volume data is used for the 3D distance field. Both volumes are loaded to the GPU memory as 3D textures. The total amount of GPU memory used for the 3D texture is 180.5 MB.

We used peak signal-to-noise ratio (PSNR) for measuring the quality of the resulting images. PSNR is computed

by Eq. 4.

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right), \tag{4}$$

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 . \tag{5}$$

MAX_I^2 is the maximum intensity value 255, $I(i, j)$ means an original image generated at the server, and $K(i, j)$ means a final image displayed on the client. For each image pair, we computed three PSNR values for the red, green and blue channels and determined the final quality factor as their average value.

Figure 8 shows the expected total time taken for rendering, encoding, transferring, decoding, and displaying one frame according to the different compression methods. In this bar chart, rendering, encoding, decoding and displaying times were measured by experiments; the transferring time was not. The transferring time is the time expected for sending a encoded frame through one MB per second transmission channel. In the wireless local area network, the practical transmission speed may vary, depending on the communication environment. Accordingly, the transmission speed may be slower than the expected one. We chose the JPEG compression method, which exhibited the best performance, for the following experiments that we used to verify our proposed methods. As shown in Fig. 8, the average PSNR of the resulting images with lossless compression methods was 39.1, which represents the maximum PSNR value that can be presented by the PDA device.

In our first experiment, we measured PSNR of the resulting images on the client for comparing the proposed method, which transfers image and vector data separately to the previous method [16], which transfers image data only. In addition, we measured the time taken for separating vector data from the image data and for displaying both data to estimate the overhead of our proposed method. In addition, we computed the average transferred data size for each rendering result. JPEG was used to compress the image data for both the previous and our proposed methods.

As shown in Table 1, our proposed method enables the client to display about 17% better quality results in PSNR than the previous image only methods. However, it requires an additional 10.5 ms per frame for separating vector data from image data and increases the average data size. Even though the transferred data size increased with our method, it is negligible under wireless LAN bandwidth. With a slightly increased overhead, our proposed method showed fairly good quality on the client. One of the main reasons that we were able to obtain better quality is that our method avoids quality degradation of the

Table 1. Performance of transferring image and vector data using our method versus image data only

Data encoding		Image & Vector	Image only
PSNR		35.7	30.5
Time	Separating & Encoding	19.6 ms	15.9 ms
	Decoding & Display	100.5 ms	96.0 ms
Average size		24 719 B	21 899 B

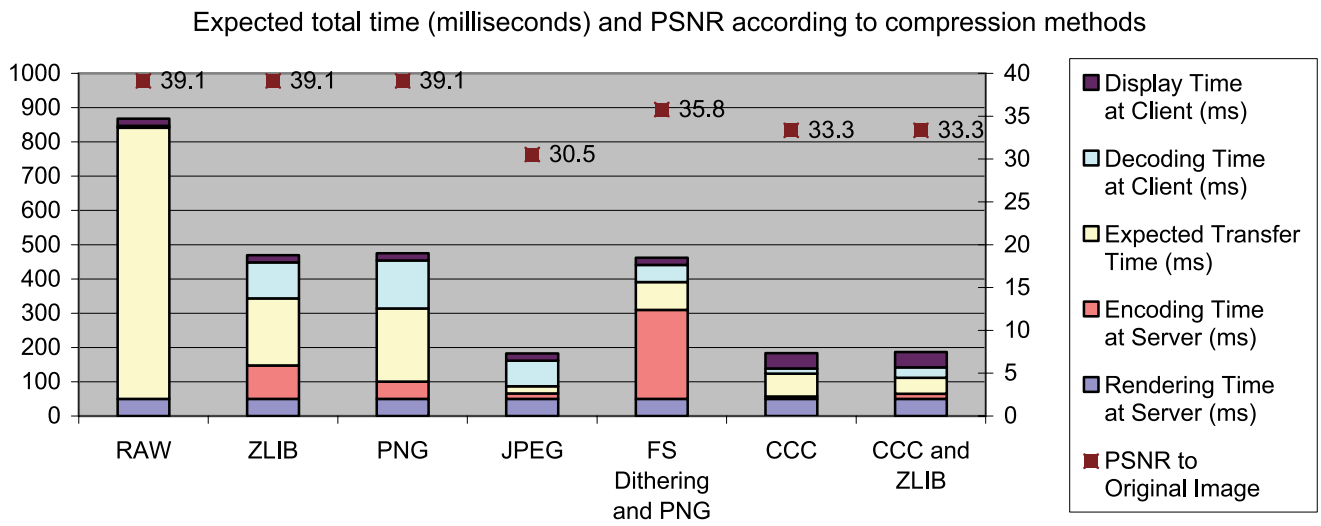

Fig. 8. Expected total time needed for rendering, encoding, transferring, decoding, and displaying one frame for different compression methods

JPEG around the boundary between colon surface and the centerline by separating the centerline data from the endoscopic view image.

In our second experiment, we simulated navigation of the colon with one thousand frames. In addition, we conducted the same navigation, varying the number of consecutive frames from 1 to 30. This experiment emphasizes that requesting a series of consecutive frames during navigation makes the frame rate higher than when requesting one frame for each navigation step.

In Fig. 9, we can see that the transmission speed depends on the number of consecutive frames for both the half frame size and the full frame size. The graph shows a sudden rise of the speed up to the value of five consecutive frames. The value of 1 represents the request of one frame for each navigation step, which is used by the previous method [16].

Figure 10 shows the frame rates on the client side according to the number of consecutive frames. The graph of the frame rates also rises dramatically up to the value of 5. For the half frame size, it continues to steadily rise up to the value of 14.

Figure 11 shows the average transmission time for each series of consecutive frames, which means the time a user should have to wait before seeing the first frame of a request after the last frame of the previous request has been displayed. It should be less than 0.5 seconds to provide reasonable interactivity. Based on the above, we determined the value of 5 as the default number of consecutive frames.

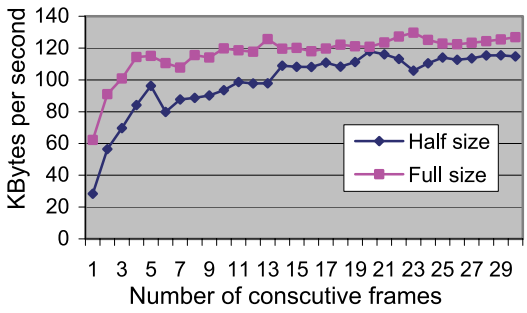

Fig. 9. Transmission speed according to the number of consecutive frames (the previous method used the value of 1); the transmission speed increases dramatically up to the number of five consecutive frames

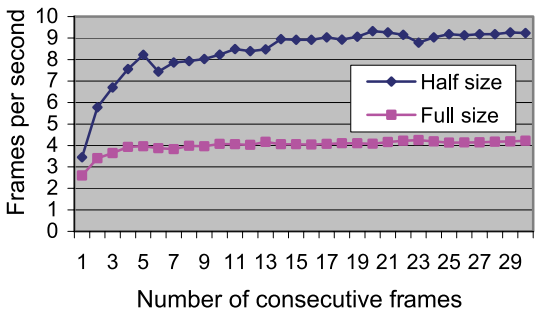

Fig. 10. Frame rates according to the number of consecutive frames (the previous method used the value of 1); the frame rates rises up to a value of 5

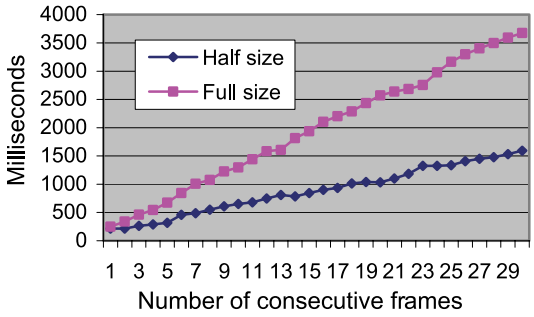

Fig. 11. Transmission time according to the number of consecutive frames (the previous method used the value of 1); the average transmission time means the time that the user has to wait before seeing the first frame of a request after the last frame of the previous request has been displayed

We also conducted the same test changing the type of image shown in the main view for the half frame size. Table 2 shows the frame rates according to the type of the main view image and the number of consecutive frames. As shown in the table, sending a series of frames for a request is on average about 2.3 times faster than sending one frame for a request when we use the value of 5 consecutive frames. However, we could get 10.5 frames per second on average when the number of consecutive frames is 20. In addition, it shows that the best frame rate is achieved when displaying the cross sectional slice image in main view.

In our third experiment, we tested how much the intermediate frame generation improves the user interactivity. To measure how many intermediate frames are generated on average, the client randomly generated ten pen moving events at different positions inside the main view, sending the rotation requests to the server for all pen events and regenerating the same events one hundred times for the

Table 2. Frame rates (fps) according to the type of the main view image and the number of consecutive frames for the half frame size

Number of Consecutive Frames	1	5	10	20
Endoscopic	3.5	8.2	8.2	9.3
Cross	3.6	8.6	10.8	11.5
Axial	3.6	8.0	9.6	10.5
Coronal	3.5	8.3	9.6	10.4
Sagittal	3.4	8.3	10.0	10.6
Overview	3.5	7.7	9.1	10.4
Average	3.5	8.2	9.6	10.5

endoscopic image and the colon surface overview image. We compared overall frame rates on the client–server side for rotation with the proposed inter-frame generation method and without it.

Table 3 shows that intermediate frame generation enables the system to display about seven times more frames to the user for the same pen moving events. In addition, the proposed method makes the overall frame rates up to 2.3 times faster than the method without intermediate frames for the rotation operation. Consequently, the test results show that the intermediate frame generation method can provide improved user interactivity.

In our fourth experiment, we measured the rendering performance of the server with GPU-based ray-casting using a 3D distance field. The rendering performance without the 3D distance field was also measured. For both cases, the server generated all navigating frames from the rectum to the cecum on the server side only.

To improve the overall performance of the client–server based visualization system, real-time rendering on the server is essential. As shown in Table 4, the endoscopic image can be rendered at 21.1 frames per second with the 3D distance field but at only 8.1 frames per second without it. Therefore, using 3D distance field for GPU-based ray-casting enables the server to render in real-time.

Table 3. Frame rates (fps) according to the type of the main view image in half frame size, with or without an intermediate frame

Frame size	Main view	Inter. Frame	# of frames	Frame rates (fps)
240 × 288	Endoscopic	without	500	3.7
		with	3250	8.5
	Colon overview	without	500	3.7
		with	3050	8.4
480 × 576	Endoscopic	without	500	2.9
		with	3700	4.0
	Colon overview	without	500	3.2
		with	3550	4.2

Table 4. Rendering performance (fps) of the GPU-based ray-casting with or without a 3D distance field

GPU-based ray-casting	With Distance field		Without Distance field	
Main view	240 × 288	480 × 576	240 × 288	480 × 576
Endoscopic	42.8	21.1	23.2	8.1
Cross	64.0	64.0	60.6	56.2
Axial	64.0	64.0	60.5	56.2
Coronal	64.0	64.0	60.4	56.0
Sagittal	64.0	64.0	59.2	53.8
Overview	64.0	64.0	60.5	53.1

8 Conclusions and future work

We have presented a wireless virtual colonoscopy system using a PDA. This system provides interactive virtual colon navigation in a wireless network environment with a mobile PDA. We proposed four methods to improve the quality of the resulting images, overall frame rates, and user interactivity. For the proposed methods, we provided experimental results to show the improvement as compared to alternative methods. First, we improved the quality of the resulting images by sending vector data and image data separately from the server to the client. This method is able to display the resulting image on the client with 17% better quality in PSNR. Second, we increased the overall frame rates by sending a series of frames for each request. Our experimental results show that sending five consecutive frames for each request makes the frame rates about 2.3 times faster than sending one frame for a request. Third, to provide better interactivity for the user during the rotation and zoom operations, the server automatically generates intermediate frames between successive event positions. The user can see about seven times more frames than when there is no automatic generation of intermediate frames. Finally, we adapted the 3D distance field to GPU-based ray-casting for rendering an endoscopic image on the server. Using the 3D distance field enables the server to render an endoscopic image about three times faster than without it. Thus, it can accelerate the overall rendering speed of our system. With the proposed methods, doctors can investigate a patients' colon by carrying a mobile device rather than sitting in front of the PC workstation.

For future work, we plan to implement our client program on a PDA equipped with a hand-held GPU, such as NVIDIA GoForce 5500, which can decompress JPEG images and support real-time standard video stream playback and the programmable pixel shader. Our current PDA takes about 113 milliseconds to decompress a full frame sized JPEG image. If we can take advantage of the graphics hardware on the PDA, we can develop more sophisticated methods utilizing image coherency. In addition, we will adapt our proposed methods to wired remote visualization systems.

Acknowledgement This research was supported by the IT National Scholarship Program supervised by IITA, the Ministry of Information and Communication, Republic of Korea and NIH grants CA082402 and CA110186. We disclose that one of the authors owns a fraction of Viatronix Inc. shares. The CT colon data sets are courtesy of Stony Brook University Hospital. The authors would like to thank Wei Hong and Feng Qui for their comments about VC and implementation of the 3D distance field.

References

1. Ang, C.S., Martin, D.C., Doyle, M.D.: Integrated control of distributed volume visaulization through the World-Wide-Web. IEEE Visualization, pp. 13–22 (1994)
2. Beaulieu, C., Jeffrey, R., Karadi, C., Paik, D., Napel, S.: Display modes for CT colonography part II: Blinded comparison of axial CT and virtual endoscopic and panoramic endoscopic volume-rendered studies. Radiology **212**, 203–212 (1999)
3. Brachtl, M., Šlajs, J., Slavík, P.: PDA based navigation system for a 3D environment. Computers and Graphics **25**(4), 627–634 (2001)
4. Campbell, G., DeFanti, T.A., Frederikson, J., Joyce, S.A., Lawrence, A.L., Lindberg, J.A., Sandin, D.J.: Two bit/pixel full color encoding. SIGGRAPH, pp. 215–223 (1986)
5. Diepstraten, J., Gorke, M., Ertl, T.: Remote line rendering for mobile devices. Computer Graphics International (CGI), pp. 454–461 (2004)
6. Engel, K., Sommer, O., Ernst, C., Ertl, T.: Remote 3D visualization using image-streaming techniques. In Advances in Intelligent Computing and Multimedia Systems, International Symposium on Intelligent Multimedia and Distance Education (ISIMADE), Baden-Baden, Germany, pp. 91–96 (1999)
7. Engel, K., Hastreiter, O., Tomandl, B., Eberhardt, K., Etrl, T.: Combining local and remote visualization techniques for interactive volume rendering in medical applications. IEEE Visualization, pp. 449–452 (2000)
8. Engel, K., Sommer, O., Ertl, T.: A framework for interactive hardware accelerated remote 3D-visualization. EG/IEEE TCVG Symposium on Visualization, VisSym, pp. 167–177 (2000)
9. Fenlon, H., Nunes, D., Schroy, P., Barish, M., Clarke, P., Ferrucci, J.: A comparison of virtual and conventional colonoscopy for the detection of colorectal polyps. New Engl. J. Med. **341**(20), 1496–1503 (1999)
10. Floyd, R.W., Steinberg L.: An adaptive algorithm for spatial greyscale. Proc. Soc. Inform. Display **17**, 75–77 (1976)
11. Gailly, J., Adler, M.: zlib home site. http://www.zlib.net (2005)
12. Hong, L., Muraki, S., Kaufman, A., Bartz, D., He, T.: Virtual voyage: Interactive navigation in the human colon. SIGGRAPH, pp. 27–34 (1997)
13. Independent JPEG Group: Library for JPEG image compression. http://www.ijg.org (1998)
14. Kaufman, A.E., Lakare, S., Kreeger, K., Bitter, I.: Virtual colonoscopy. Commun. ACM **48**(2), 37–41 (2005)
15. Krüger, J., Westermann, R.: Acceleration techniques for GPU-based volume rendering. IEEE Visualization, pp. 287–292 (2003)
16. Lamberti, F., Zunino, C., Sanna, A., Fiume, A., Maniezzo, M.: An accelerated remote graphics architecture for PDAs. ACM/SIGGRAPH Web3D Symp., pp. 55–61 (2003)
17. Lamberti, F., Sanna, A.: A solution for displaying data models on mobile devices. WSEAS Trans. Inform. Sci. Applic. **2**(2), 258–264 (2005)
18. Lippert, L., Gross, M.H., Kurmann, C.: Compression domain volume rendering for distributed environments. EUROGRAPHICS, pp. 95–107 (1996)
19. Maurer, C.R., Qi, R., Raghavan, V.: A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. IEEE PAMI **25**(2), 265–270 (2003)
20. Microsoft Corporation: Windows Mobile 5.0 SDK for Pocket PC. http://msdn.microsoft.com/mobility/downloads/sdks/default.aspx (2006)
21. NVIDIA Corporation: Cg Toolkit 1.4.1. http://developer.nvidia.com/object/cg_toolkit.html (2006)
22. Pickhardt, P.J., Choi, R., Hwang, I., Butler, J.A., Puckett, M.L., Hildebrandt, H.A., Wong, R.K., Nugent, P.A., Mysliwiec, P.A., Schindler, W.R.: Computed tomographic virtual colonoscopy to screen for colorectal neoplasia in asymptomatic adults. New Engl. J. Med. **349**(23), 2191–2200 (2003)
23. Quillet, J., Thomas, G., Granier, X., Guitton, P., Marvie, J.: Using expressive rendering for remote visualization of large city models. 3D technologies for the WWW, Proceedings of the 11th Int. Conf. on 3D Web Tech., pp. 27–35. ACM Press (2006)
24. Rizzo, F., Storer, J.A.: Overlap in adaptive vector quantization. Data Compression Conference, pp. 401–410. IEEE, Snowbird, UT, USA (2001)
25. Schalnat, G.E., Dilger, A., Randers-Pehrson, G.: Official PNG reference library. http://www.libpng.org (2006)
26. Stegmaier, S., Diepstraten, J., Weiler, M., Ertl, T.: Widening the remote visualization bottleneck. Proceedings of Image and Signal Processing and Analysis (ISPA 2003), pp. 174–179. IEEE (2003)
27. Stegmaier, S., Strengert, M., Klein, T., Ertl, T.: A simple and flexible volume rendering framework for graphics hardware-based raycasting. Int. Workshop for Volume Graphics, pp. 187–195. IEEE/EG, Stony Brook, NY (2005)
28. Wan, M., Tang, Q., Kaufman, A., Liang, Z.: Volume rendering based interactive navigation within the human colon. IEEE Visualization, pp. 397–400 (1999)
29. Yee, J., Akerkar, G., Hung, R., Steinauer-Gebauer, A., Wall, S., McQuaid, K.: Colorectal neoplasia: Performance characteristics of CT colonography for detection in 300 patients. Radiology **219**, 685–692 (2001)

SEOK-JAE JEONG is currently a postdoctoral associate at the Center of Visual Computing (CVC) at the State University of New York at Stony Brook. He received a BE degree, an ME degree and a PhD in computer engineering from Ajou University in 1996, 1998, and 2004, respectively. His research interests include volume rendering, remote visualization, pattern matching, mobile computing, and ubiquitous computing.

ARIE E. KAUFMAN is Chairman of the Computer Science Department, the Director of the Center of Visual Computing (CVC), and a Distinguished Professor of Computer Science and Radiology at the State University of New York at Stony Brook. Professor Kaufman has conducted research for over 30 years in computer graphics and specifically computer graphics architectures, algorithms, and languages; visualization including volume visualization and information visualization; user interfaces; virtual reality; and multimedia; with specific applications in biomedicine. He has lectured widely and published more than 250 technical papers in these areas, including the IEEE tutorial book on volume visualization, and he holds or has filed more than 30 patents, many of which have been licensed. Professor Kaufman is a Fellow of IEEE. He was the founding Editor-in-Chief (1995–1998) of IEEE Transactions on Visualization and Computer Graphics (TVCG) and has been an Editor of Computer Graphics Forum, Computers and Graphics, The Visual Computer, and Computer Graphics and Applications. He was Chair for the international workshops on volume graphics (2001, 2003, 2005), co-Chair for Eurographics/SIGGRAPH graphic hardware workshops (1990, 1991, 1997, 1998, 1999), program co-chair for the symposia on volume visualization (1992, 1994, 1998), papers/program co-chair for "Visualization" (1990–94), and co-founder and steering committee member of the visualization conference series. He has previously chaired and is currently a director of IEEE CS Technical Committee on Computer Graphics. He is the recipient of the 1995 IEEE CS Outstanding Contribution Award, the 1996 IEEE CS Golden Core Member, the 1998 ACM Service Award, the 1999 IEEE CS Meritorious Service Award, the 2002 NYS Entrepreneur Award, the 2002 NYS Entrepreneur Award, the 2004 IEEE Harold Wheeler Award, and the 2005 SUNY Chancellor's Technology Transfer Award. He received a BS in mathematics and physics from the Hebrew University of Jerusalem in 1969, an MS in computer science from the Weizmann Institute of Science, Rehovot, in 1973, and a PhD in computer science from the Ben-Gurion University, Israel, in 1977.