Optica Applicata, Vol. XXXVIII, No. 3, 2008

An iterative programmable graphics process unit based on ray casting approach for virtual endoscopy system

FEINIU YUAN

School of Information Technology, Jiangxi University of Finance and Economics, Nanchang 330013, Jiangxi, China; e-mail: yfn@ustc.edu

State Key Lab of Fire Science, University of Science and Technology of China, Hefei 230027, Anhui, China

In this paper, a fast graphics process unit (GPU) based ray casting algorithm is presented to improve image quality. A linear interpolation is used to estimate the intersection between a ray and isosurfaces. Thus, resampling artefacts is greatly reduced and the performance is not influenced. An iterative estimation is presented to further improve image quality. According to the distance the ray goes across, *z* values in the *z*-buffer are modified to implement hiding of hybrid scenes. Experimental results show that the algorithm can produce high quality images at interactive frame rates and implement hiding of hybrid scenes very well.

Keywords: virtual endoscopy, linear interpolation, medical image processing, graphics process unit (GPU), ray casting, isosurface.

1. Introduction

Virtual endoscopy is a new diagnostic and surgery planning method that is non-invasive and reusable compared with traditional optical endoscopy. There are two main groups of visualization techniques including surface rendering and volume rendering. In surface rendering techniques, intermediate polygons of isosurfaces must be extracted from the 3D volumetric data set and then the polygons are rendered using the traditional computer graphics rendering algorithms. It can obtain interactive rendering rates. However, the quality of the rendered images is not very high because of much detailed information lost during the process of extraction of polygons. While in volume rendering techniques, extraction of polygons is not required and the volume data set is directly rendered according to a specific transfer function, so it can produce high quality images. But it is memory and computation intensive. Ray casting algorithm is a volume rendering technique, which belongs to image space rendering techniques.

It can generate high quality images and often be applied in virtual endoscopy systems. It can obtain high frame rates on high end workstations and special volume rendering hardware (such as VolumePro [1], *etc*.) and so on. However, it is unable to acquire interactive frame rates on the popular PC platform without special purpose hardware.

There are many acceleration techniques to speed up the brute force ray casting algorithm. One kind of these techniques is the acceleration technique based on space leaping, which can avoid many empty samplings and does not degrade the image quality, such as cylindrical approximation of tubular organs presented by VILANOVA *et al.* [2], spherical approximation of tubular organs proposed by SHARGHI and RICKETTS [3]. Another kind of these techniques is the acceleration technique with a fundamental tradeoff between image quality and rendering speed, such as two-phase perspective ray casting for interactive volume navigation presented by BRADY *et al*. [4], screen and object adaptive sampling, *etc*.

A third kind of acceleration techniques is based on consumer level graphics hardware. There are two distinct approaches.

The first approach which originally presented by CULLIP *et al*. [5] and further developed by CABRAL *et al*. [6] is directly exploiting the texture mapping capabilities of graphics hardware by resampling proxy surfaces. Proxy surfaces are either axis--aligned [7] with three 2D textures or view-aligned [8] with one 3D texture, as shown in Fig. 1. This technique can obtain interactive frame rates, but produce relatively low quality images.

The second approach is to implement a ray caster in the fragment shader of the GPU, as proposed by KRUGER and WESTERMANN [9]. In the algorithm, the dataset is stored as a 3D-texture to take advantage of the built-in tri-linear interpolation in graphics hardware [10]. And then a bounding box for the dataset is created where all the coordinates of 8 corner points are within the range from 0.0 to 1.0. Supposed each corner point (*x*, *y*, *z*) of the bounding box has the color (*r*, *g*, *b*), and then *r*, *g*, *b* are equal to *x*, *y*, *z*. That is to say, the position on the surface of the box is encoded in the color channel, as shown in the Fig. 2**a**. The viewing vector at any given pixel can be easily computed by subtracting the color of the front surfaces of the color cube at this pixel from the color of the back surfaces at this pixel, as shown in Fig. 2. The color of the front surfaces is also regarded as the start point at each pixel for the ray casting process. And the color of the back surfaces is regarded as the end point at each pixel for the ray casting process. KIM *et al*. [11] presented vertex transformation streams based on GPU, which addressed the input geometry bandwidth bottleneck for interactive 3D graphics applications. Flexibility of GPU programming improves parallel computing performance in many time-critical applications [12, 13].

In this paper, an optimized approach is presented, in order to avoid artifacts and improve rendering performance. Modification of *z* values in the fragment shader program can simply implement hybrid visualization of polygons and volumetric objects. Our approach is similar to the algorithm that NEUBAUER *et al*. [14] presented. But our approach is different from Neubauer's method in three aspects. First, the approach is based on GPU and it can obtain interactive frame rates. Second, it uses

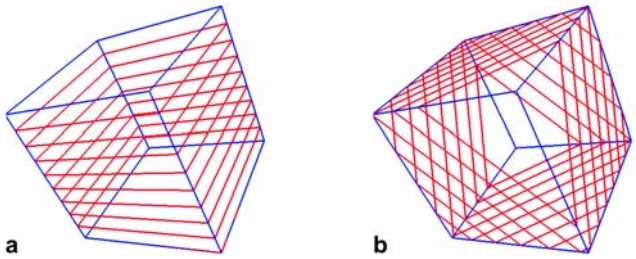

Fig. 1. Axis-aligned (**a**) and view-aligned resampling slices (**b**).

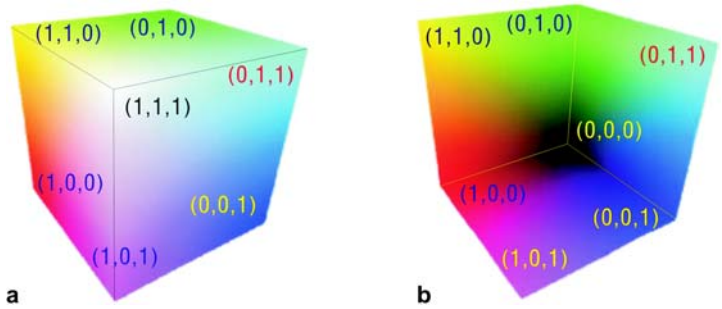

Fig. 2. Front (**a**) and back (**b**) faces of the bounding box encoding the position.

regular ray traversal strategy with linear estimation of intersection, so it does not need any special data structure. Third, *z* modification implements hybrid visualization.

This paper is organized as follows. In Section 2, an initial plane is created to generate start points of all the rays. Section 3 discusses the iterative linear interpolation of intersection to avoid artifacts and speedup rendering. In Section 4, modification of *z* values is used to implement hiding of polygons and volumes. At last, some experiments are performed and conclusions are drawn.

2. Entering into the volume

For virtual endoscopy applications, the viewpoint is usually located in the volume to generate similar views to those produced by traditional optical endoscopic diagnoses. In the GPU based ray casting algorithm presented by KRUGER and WESTERMANN [9], front and back surfaces of the bounding box of the volume are successively rendered to produce entry and exit points of the ray cast from the viewpoint. However, when the viewpoint moves into the volume, the front surfaces of the bounding box would become invisible. In this case, the color of the front surfaces usually becomes solid color and does not contain any position information. So, we cannot directly compute the viewing vector by subtracting the color of the front surfaces from the color of the back surfaces. In order to solve this problem, when the virtual camera is located in the volume, an initial plane is created in the neighborhood of the near clipping plane

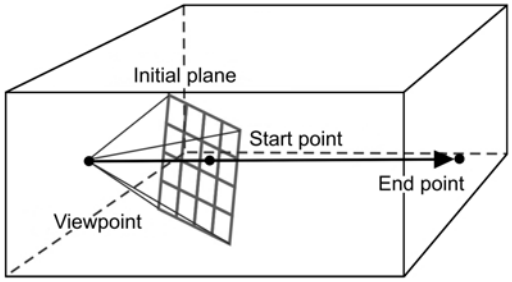

Fig. 3. The viewpoint in the volume.

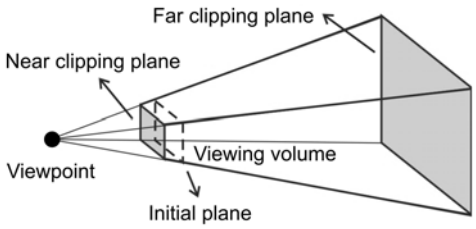

Fig. 4. Location of the initial plane.

that replaces the front surface of the bounding box as the proxy geometry where start points of all the rays are located in, as shown in Fig. 3. And end points of all the rays are still located in the back surfaces of the bounding box.

The initial plane is parallel to the near clipping plane and located in the viewing volume to assure that the initial plane is visible all the time, as shown in Fig. 4. And the initial plane is very close to the near clipping plane. Following the processes

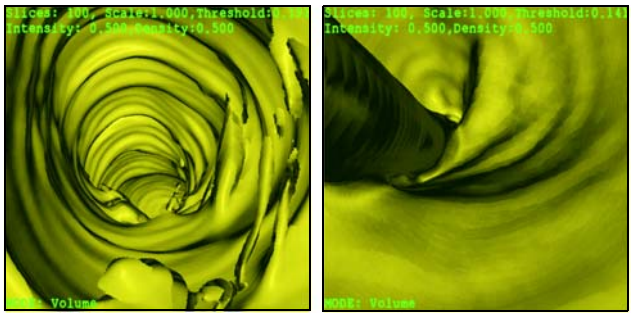

Fig. 5. Virtual endoscopic views.

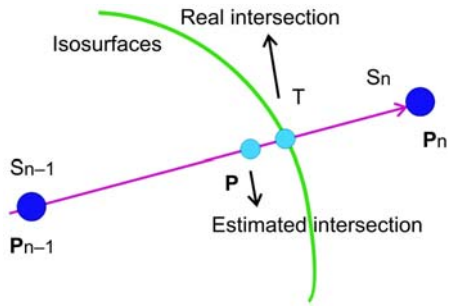

Fig. 6. Linear interpolation of intersection.

mentioned above, the viewing ray can be directly computed by subtracting the color of the initial plane from the color of the back surfaces of the bounding box.

The GPU based ray casting algorithm for virtual endoscopic applications is described as follows:

1. Render back surfaces of the bounding box into an intermediate texture.

2. Render the initial plane, subtract the color of initial plane from the color of back surfaces to get a direction vector, store the normalized vector together with the length of the vector in a direction texture. The length of the vector is just the maximum distance that the ray goes across. The normalized viewing vector and the distance are encoded in the color channel and the alpha channel, respectively.

3. Regard the color of initial plane as an inputted start point of the ray for the fragment shader program. Cast a ray along the viewing vector stored in the direction texture, and composite color and opacity of each resampling. Terminate the ray if the distance the ray goes across is greater than the distance stored in alpha channel of the direction texture or the opacity has reached a certain threshold (early ray termination).

Figure 5 illustrates two experimental results by the algorithm. From rendered images, we can find that colon and chine cavity of human are clearly displayed by GPU based ray casting algorithm.

3. Linear interpolation of intersection

In many virtual endoscopic diagnoses, rendering of one or multiple isosurfaces is usually adopted. High accurate computation of intersection between the ray and isosurfaces is very important for correct rendering. But high accurate computation of intersection will greatly increase the searching time of isosurfaces and markedly decrease frame rates.

In order to improve the performance, a linear interpolation of intersection is proposed to approximate the actual intersection. As shown in Fig. 6, s_{n-1} and $\mathbf{P}_{n-1}$ are the intensity value and position of resampling at the $(n-1)$-th step, respectively; s_n and $\mathbf{P}_n$ are the intensity value and position of resampling at the n-th step, respectively. T is a threshold value for an isosurface and $\mathbf{P}$ is the estimated intersection between the ray and isosurface. When s_{n-1} is less than T and s_n is greater than T, there must be an isosurface between these two resampling points. So, the estimated intersection $\mathbf{P}$ can be computed by the following equation:

$$\mathbf{P} = \frac{s_n - T}{s_n - s_{n-1}} \mathbf{P}_{n-1} + \frac{T - s_{n-1}}{s_n - s_{n-1}} \mathbf{P}_n \tag{1}$$

where

$$\mathbf{P}_n = \mathbf{P}_0 + \mathbf{L}\, d_n \tag{2}$$

$$d_n = n\, d \tag{3}$$

In the above Eqs. (1)–(3), d is the resampling step size, $\mathbf{P}_0$ – the start point in the initial plane, $\mathbf{L}$ – the normalized vector of a ray and n the count of steps.

According to the estimated intersection, the Phong lighting effects are computed for current pixel. Linear interpolation of intersection with a relatively larger resampling distance can improve frame rates without obviously degrading image quality. Searching codes of isosurfaces in the fragment shader program can be written as follows:

```
   /* P0 is the start point in the initial plane. DirectionTexture is
the direction texture containing the normalized viewing ray and the
maximum distance the ray goes across */
   vec4 L=texture3D(DirectionTexture, P0);
   vec3 Pn, P;
   vec4 vol;
   float dn=0.0;
   float dp;
   float sn, sn_1=0.0;
   while (dn<L.a)
   { Pn =P0 +L.xyz*dn;
     vol=texture3D(VolumeTexture, Pn);
     //T is the threshold for isosurfaces
     if(vol.a<T)
     { sn_1=vol.a;
      dn=dn+d; //d is the resampling step size
     }else
     { sn=vol.a;
      break;
     }
   }
   if(dn>=L.a) discard;
   if((sn- sn_1)>=0.0001)
   {
     dp=dn-d*(sn-T)/ (sn- sn_1);
   }else
   {
     dp=dn;
   }
   P=P0+L.xyz*dp; //P is the estimated intersection
```

After the above searching with linear interpolation, the estimated intersection is reasonably accurate. So, using the estimated intersection, we can produce high quality images. We can repeat the linear interpolation of intersection again using newly estimated positions $\mathbf{P}$ and $\mathbf{P}_n$ or $\mathbf{P}_{n-1}$, in order to obtain more accurate estimation of intersection between the ray and isosurfaces. Experiments show that only 1 to 3 times are enough to obtain very good estimation of intersection. So, it is still useful for acceleration of rendering. Iterative searching codes in the fragment shader program are given as follows:

```
   int i=0;
   float sp,dn_1;
   dn_1=dn-d;
   while(i<3)
```

```
{ vol=texture3D(VolumeTexture, P);
  sp=vol.a;
  /*When the estimation error is very small, iteration can be stopped
  in advance*/
  if(abs(sp-T)<0.0001) break;
  if(sp<T)
{ /*The resampling step d is the half distance between points P and
  Pn */
  d=(dn-dp)/2.0;
  sn_1=sp;
  dn_1=dp;
}else
{ /*The resampling step d is the half distance between points Pn_1
  and P */
  d=(dp-dn_1)/2.0;
  sn=sp;
  dn=dp;
}
dp=dn_1+d;
P=P0 +L.xyz*dp;
vol=texture3D(VolumeTexture, P);
if(vol.a>T)
{ //Linear interpolation between Pn_1 and P
  sn=vol.a;
  dp=dp-d*(sn-T)/ (sn- sn_1);
}else
{ //Linear interpolation between P and Pn
  sn_1=vol.a;
  dp=dn-d*(sn-T)/ (sn- sn_1);
}
//P is the estimated intersection at current time
  P=P0+L.xyz*dp;
  i=i+1;
}
```

In the above iterative program, if the estimation error is very small, for example, it is less than 0.0001, the iteration can be early terminated in order to improve performance, otherwise, the linear estimation must be performed for this time. And then, if the resampling value at the estimated position **P** is less than the threshold *T*, the linear interpolation of intersection should be performed between positions **P** and $\mathbf{P}_n$.

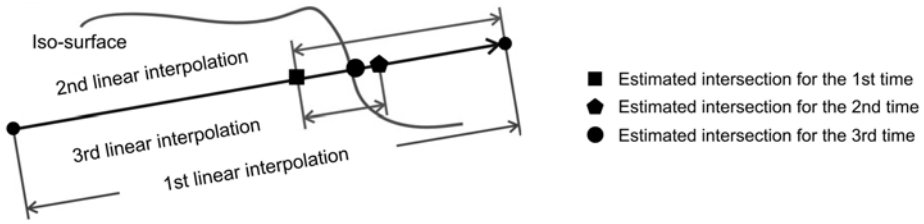

Fig. 7. Iterative linear interpolation of intersection.

Accordingly, the step d is the half distance between $\mathbf{P}$ and $\mathbf{P}_n$. Otherwise, the linear interpolation should be done between points $\mathbf{P}_{n-1}$ and $\mathbf{P}$, and the step d is set to the half distance between $\mathbf{P}_{n-1}$ and $\mathbf{P}$.

The iterative process can be shown as Fig. 7. In Figure 7, a quadrangle stands for the estimated intersection for the first time, a pentagon for the second time, and a circle for the third time. As we can see, the accuracy is rapidly improved while times of iteration increase.

4. Hybrid visualization

In virtual endoscopy systems, volumetric datasets and traditional polygons usually exist together. Traditional primitives often assist doctors to complete the diagnosis, for example, polygons may be used to indicate the orientation and location of the virtual camera in the 3D datasets. So, the problem must be solved of how to correctly render the hybrid scenes.

In the *z*-buffering algorithm, hiding is naturally implemented according to depths of screen pixels and the principle is very simple. So, it is very easy to be implemented in hardware. Nearly all the consumer-level graphics hardware supports *z*-buffering. Due to the mathematics involved, the generated value z in a *z*-buffer is not distributed evenly across the *z*-buffer range (typically, 0.0 to 1.0, inclusive)

$$z = \frac{z_f}{z_f - z_n}\left(1 - \frac{z_n}{r}\right) \tag{4}$$

As shown in Eq. (4), r is the actual distance between the viewpoint and the estimated intersection, z_n is the distance between the viewpoint and near clipping plane, and z_f is the distance between the viewpoint and far clipping plane. Corresponding z value in the *z*-buffer can be computed according to Eq. (4). So, in the fragment shader program, we add the following codes before writing the color to frame buffer:

```
float z=10.0/(10.0-0.002)*(1.0-0.002/r);
if(z<gl_FragDepth) discard;
gl_FragDepth=z;
```

In our implementation, z_n and z_f are equal to 0.002 and 10.0, respectively, and the variable r is the actual distance between the viewpoint and the estimated intersection. In Figure 8, the hybrid scene includes an MRI head volume and polygons of the bounding box. Figure 8**a** illustrates no hiding effects and Fig. 8**b** shows the hiding effects after adding the above codes in the fragment shader program. We can see that modification of z values can visualize the hybrid scene correctly.

5. Experiments

We have implemented the proposed and traditional GPU based ray casting algorithms using VC++ and OpenGL, and several experiments were performed on a Pentium D/ 3.0GHz PC with a GeForce FX7300 graphics card. Compared with traditional 3D

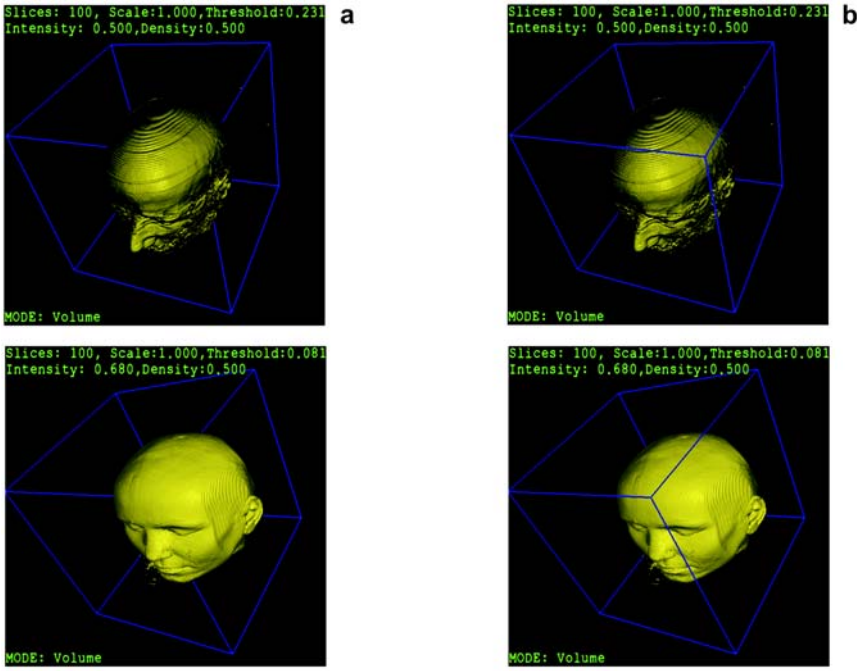

Fig. 8. Visualization of hybrid scene including an MRI volume and its bounding box, **a** – no hiding, **b** – hiding.

texture based methods, our optimized method produces high quality images at interactive frame rates.

First, navigate into an abdomen volumetric dataset (512×512×86) with the algorithm proposed. As shown in Fig. 9, we can find that the rendered images have high quality. Those images were generated using the isosurface ray casting based on GPU. We can

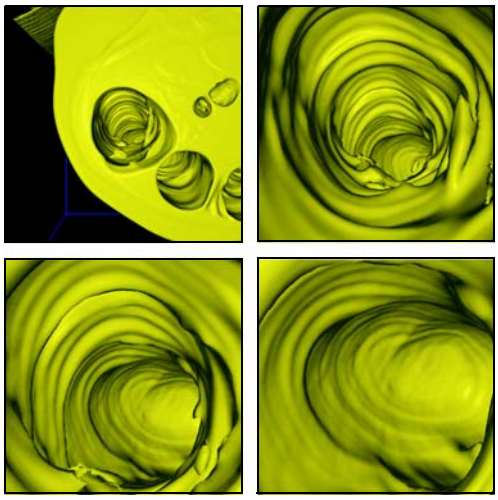

Fig. 9. Rendered images by the algorithm while interactively navigating through human colon.

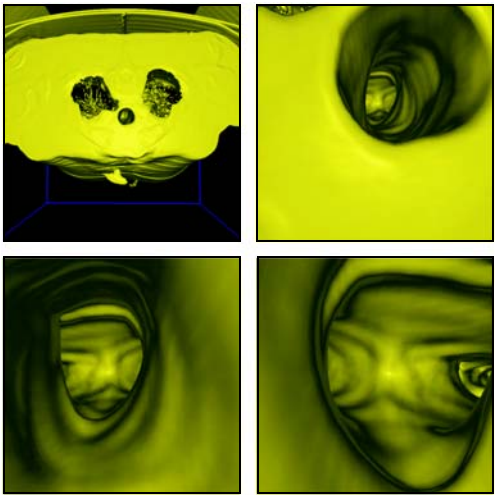

Fig. 10. A series of images rendered while interactively navigating through human trachea.

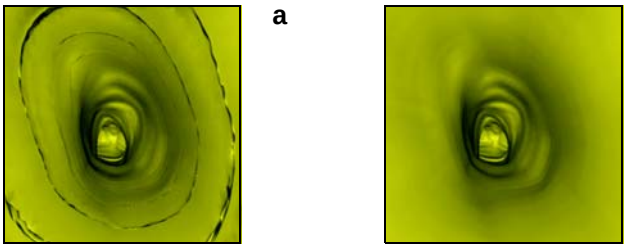

Fig. 11. A comparison of image quality: **a** – traditional method, **b** – our method.

observe highlight effects on the colon due to enabling the Phong illumination model after classification.

Second, navigate into another volumetric dataset (512×512×112) with the algorithm. As shown in Fig. 10, we can find that the rendered images have high quality. We can interactively alter the transfer function by simply regenerating the 2D lookup texture or directly specifying different ambient, diffuse and specular colors. So, our algorithm is very convenient in cases where users need to frequently modify material property.

We also implemented traditional GPU based ray casting algorithm. The Table gives frame rates. In the Table, our optimized searching approach can greatly accelerate rendering speed without degrading image quality when the resampling step in the optimized algorithm is 4 times the step in traditional algorithms. As shown in Fig. 11**a**, although large steps can accelerate rendering speed, the traditional algorithm

T a b l e. Frame rates.

Datasets	Traditional GPU ray casting	Our optimized GPU ray casting
Human colon	12.5 fps	30.2 fps
Human trachea	16.8 fps	39.3 fps

produces severe artifacts due to large steps. As shown in Fig. 11**b**, our method is able to avoid such artifacts and obtain obvious improvements of computation. In order to further speedup visualization, space leaping based acceleration techniques is very suitable for being implemented using GPU shader programs.

6. Conclusions

GPU based ray casting in the fragment shader program has been presented that allows both orthogonal and perspective projection and enables the user to move the viewpoint into the dataset for virtual endoscopic visualization. An optimized GPU based ray casting algorithm is used in virtual endoscopy applications. To avoid artifacts, a linear interpolation of intersection is presented to estimate the actual intersection between the viewing ray and the isosurface. Thus, resampling artifacts is greatly avoided and the performance is not influenced. A larger resampling step with the linear interpolation of intersection can speed up rendering. Using the accurate estimated intersection, we can produce high quality images. Iterative linear interpolation of intersection is presented in order to obtain more accurate estimation of intersection between the ray and isosurfaces. Experiments show that only 1 to 3 times are enough to obtain very good estimation of intersection. So, it is still useful for acceleration of rendering. The author takes advantage of currently available programmable graphics hardware and OpenGL Shading Language to enable Phong lighting model with a point light source on the fly. Thus, it is possible that we can acquire high quality rendered images as well as interactive frame rates. The *z* values in the *z*-buffer are modified to implement visualization of hybrid scenes that usually exist in virtual endoscopy application. Several experiments show that the optimized GPU based ray casting algorithm is very useful for virtual endoscopy.

Acknowledgements – This project was supported by Natural Science Foundation of Jiangxi Province (2007GQS0076), Foundation of Education Department of Jiangxi Province (2007[272]), Key project of Jiangxi University of Finance and Economics, Open Foundation of the State Key Lab of Fire Science (HZ2006-KF03) and China Postdoctoral Science Foundation (20070410792).

References

[1] PFISTER H., *Architectures for real-time volume rendering*, Future Generation Computer Systems **15**(1), 1999, pp. 1–9.

[2] VILANOVA I BARTROLÍ A., KÖNG A., GRÖLLER M.E., *Cylindrical Approximation of Tubular Organs for Virtual Endoscopy*, Technical Report TR-186-2-00-02, February 2000, Abteilung für Computergraphik, Institut für Computergraphik und Algorithmen, Technische Universität Wien, Austria.

[3] SHARGHI M., RICKETTS I.W., *A novel method for accelerating the visualization process used in virtual colonoscopy*, Proceedings of Fifth International Conference on Information Visualisation, 2001, San Diego, California, October 22–23, 2001, pp. 167–72.

[4] BRADY M., JUNG K., NGUYEN H.T., NGUYEN T., *Two-phase perspective ray casting for interactive volume navigation*, IEEE Visualization'97 Conference, Phoenix, Arizona, October 19–24, 1997, pp. 183–9.

[5] CULLIP T., NEUMANN U., *Accelerating volume reconstruction with 3D texture mapping hardware*, Technical Report TR93-027, Department of Computer Science, University of North Carolina, Chapel Hill, 1993.
[6] CABRAL B., CAM N., FORAN J., *Accelerated volume rendering and tomographic reconstruction using texture mapping hardware*, Proceedings of IEEE Symposium on Volume Visualization 1994, pp. 91–8.
[7] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T., *Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization*, [In] *Proceedings of Graphics Hardware 2000*, pp. 109–18.
[8] WESTERMANN R., ERTL T., *Efficiently using graphics hardware in volume rendering applications*, [In] *Proceedings of SIGGRAPH'98*, 1998, pp. 169–78.
[9] KRUGER J., WESTERMANN R., *Acceleration techniques for GPU-based volume rendering*, IEEE Visualization 2003, pp. 287–92.
[10] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T., *A simple and flexible volume rendering framework for graphics-hardware based raycasting*, Fourth International Workshop on Volume Graphics, 2005, pp. 187–241.
[11] YOUNGMIN KIM, CHANG HA LEE, AMITABH VARSHNEY, *Vertex–transformation streams*, Graphical Models **68**(), 2006, pp. 371–83.
[12] FIALKA O., CADK M., *FFT and Convolution Performance in Image Filtering on GPU*, Tenth International Conference on Information Visualization, 2006, pp. 609–14.
[13] KRUGER J., WESTERMANN R., *Linear Algebra Operators for GPU Implementation of Numerical Algorithms*, SIGGRAPH 2003, pp. 908–16.
[14] NEUBAUER A., MROZ L., HAUSER H., WEGENKITTLE R., *Cell-based first-hit ray casting*, Proceedings of the Symposium on Data, Visualization 2003, 2002, pp. 77–86.

Received December 11, 2007
in revised form January 22, 2008