Fast Volume Rendering for Medical Image Data

Cheng-Sheng Hung, Chien-Feng Huang, Ming Ouhyoung
Communication and Multimedia Laboratory
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

Abstract

Volume rendering is a technique for visualizing 3D images from 2D sampled data. For 2D medical image data, volume rendering can be used to construct the corresponding 3D images, and physicians can then use them for more precise diagnoses. In this paper, we use some speed-up methods to fast construct the 3D images. After the construction, one can use our system for real-time rotation and simulated surgery of the 3D images. The data used by our system are from the human body MRI provided by National Yang-Ming University. The goal of this project is to build the "visible human" database in Taiwan. On a Pentium II 300 PC, the speed in rotating a 3D image is 1.14 seconds per frame from 55 slices of 256x256 "full body" MRI images.

1. Introduction

In constructing 3D medical images, one often uses volume rendering for 3D reconstruction. Using this technique, one can generate 3D images from a stack of 2D medical images such as CT or MRI data to obtain better understanding of them. Volume rendering is capable of making very effective visualization, but its speed may be too slow because of its high computational complexity. In the past, many efforts have been proposed to reduce its computational cost [1,4,9]. The methods proposed to accelerate volume rendering are either in the image order such as ray tracing [2,5] or in the object order such as splatting [3]. In addition, some compact storage mechanisms have been proposed to reduce the volume size such as semi-boundary (SB) [4] and run-length coding [1]. Although these methods can efficiently reduce the volume size, their data structures may be more complicated. In this paper, we adopt ray tracing and use some simplified speed-up methods to fast construct the 3D image. Users can then fast rotate the image to observe it in any viewing angle. In addition, users can have simulated surgery on the image. By cutting the surface of any portion, users can see the inside structure of the body, and can have more information for further diagnoses.

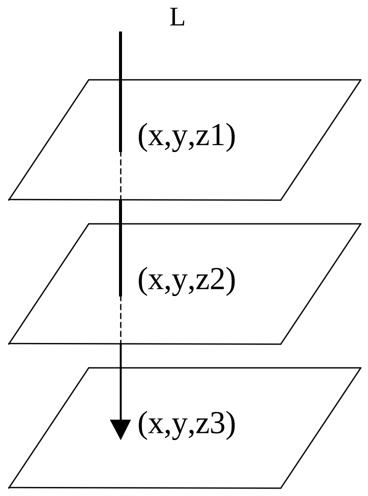
The medical image data used by the system is MRI (Magnetic Resonance Imaging) data provided by National Yang-Ming University. This is perhaps the first full human body medical image data in Taiwan. On a Pentium II 300 PC, our system can fast construct the 3D image, and rotate it in about 1 second. So our system may be more useful in real applications.

2. System Architecture and Speed-up Methods

2.1 Ray Tracing

We use ray tracing to implement volume rendering. For every pixel on the 2D images, a virtual light is cast orthogonally through it. The color of the corresponding pixel on the 3D image is the sum of color values of all the pixels cast by the same virtual light ¡ G

$$\textbf{FinalColor}(x_i, y_j) = \sum_{k=0}^{N} [\text{Color}(x_i, y_j, z_k) * \text{Opacity}(x_i, y_j, z_k) * \prod_{m=k+1}^{N} (1 - \text{Opacity}(x_i, y_j, z_m))]$$



c1,c2,c3 : the pixel color of point (x,y,z1),(x,y,z2),(x,y,z3)
a1,a2,a3 : the opacity of point (x,y,z1),(x,y,z2),(x,y,z3)

The sum of color values of all the pixels cast by light L is:

$$c1*a1 + c2*a2(1-a1) + c3*a3(1-a1)(1-a2)$$

In calculating the opacity of every pixel, the complexity is very high using the previous method [6]. First, the gradient of every pixel must be known:

$$Ñf(X_i) = Ñf(x_i, y_j, z_k) » [1/2[f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)], 1/2[f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)], 1/2[f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})]]$$

After acquiring the gradient, the opacity is calculated by linear interpolation and scaled by the magnitude of the gradient to enhance the opacity of the bounding surfaces:

$$a(X_i) = |Ñf(X_i)|\{a_{v_{n+1}}[\frac{f(X_i) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}}] + a_{v_n}[\frac{f_{v_{n+1}} - f(X_i)}{f_{v_{n+1}} - f_{v_n}}]\}$$

if $f_{v_n} \leq f(Xi) \leq f_{v_{n+1}}$

or

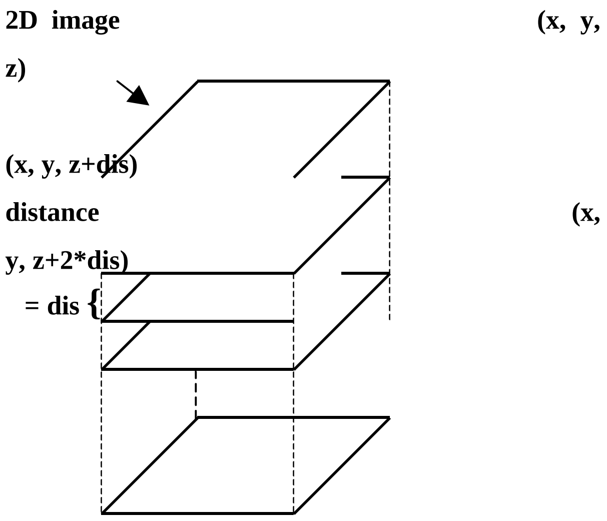
$a(X_i) = 0$, otherwise

The complexity is high because we must calculate the gradient and then use linear interpolation for every pixel. To simplify the computation of opacity, the color value of the pixel is used directly as the opacity of this pixel after scaling it down to between 0.0 and 1.0. Comparing the image constructed by this simplified method with the image constructed by the original method (see figure 1), we can make a conclusion that this new method is feasible.

After finding out the opacity, the function listed above can be used to calculate the color value of every pixel. Observing this function, we know that if the pixel is transparent (its opacity is close to 0), then we can skip it. In the same way, if the pixel is opaque (its opacity is close to 1), then all the pixels behind it will be obscured, and ray tracing can then be early-terminated. With these two checks, the computational complexity will be reduced significantly.

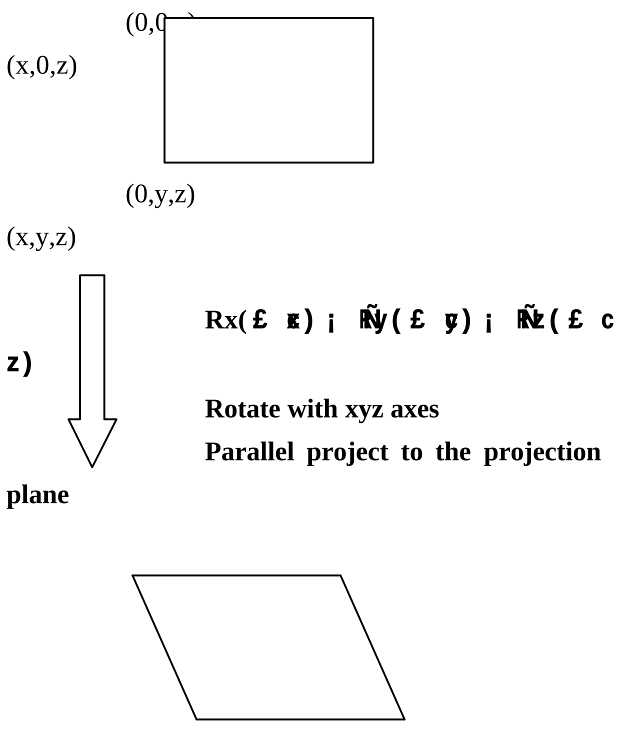
2.2 Fast Rotation On XYZ Axes

To rotate the image, the 3D coordinates of all of its pixels must be obtained first, and we can then apply the 4x4 rotation matrix to them. To obtain 3D coordinates, we first create a bounding box, and place all the 2D images into it in parallel. After setting up the distance between two images, all the Z values are obtained.



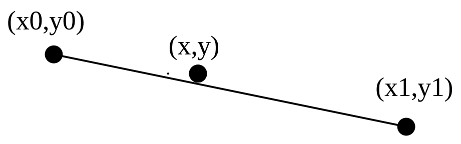
In multiplying the rotation matrix, the computational complexity will be very high if every pixel is applied to the matrix (256*256*55 pixels). To reduce the complexity, the following method is used: For every image, a horizontal scan line passes through it one row at a time. For each scan line, only the first and the last points are rotated, and then projected in parallel on the drawing plane. For the other points in the scan line, we can simply linearly interpolate these two points to find out their 2D coordinates in the drawing plane. In addition, before the linear interpolation, we check that if all of the two points fall into the drawing plane. If they are all out of the drawing plane, just skip the interpolation and go on to the next scan line. In calculating the interpolation, if the interpolated point is already out of the drawing plane, skip the following interpolation and jump to the next scan line in the same way.

In the following, we explain why this method works. After applying rotation and parallel projection on a 3D rectangle, it will become a parallelogram on the projection plane. The parallelogram is consisted of many scan lines, so we can map every scan line in the parallelogram to the corresponding one in 3D rectangle. With this simple mapping, we can simply use linear interpolation to achieve the function like texture mapping.


(0,0,z)
(x,0,z)
(0,y,z)
(x,y,z)
Rx(£ x) ¡ Ñy(£ y) ¡ Ñz(£ z)
Rotate with xyz axes
Parallel project to the projection plane

The result is a parallelogram

For every pixel on the 2D scan lines, we must find the corresponding 3D point to obtain its color value. The computation time may be slowed down here because we must first calculate the slopes of 2D scan lines. To simplify the computation, we use the reverse method to accomplish it. That is, for every point on the 3D scan lines, find the corresponding 2D pixel on the drawing plane. With this method, only linear interpolation is used, and much computation time is saved. The algorithm of our new method is listed below:



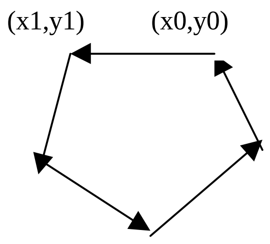
```
For each scanline j
    For each point of scanline
        X=(X0*(256-I)+X1*I)>>8
        Y=(y0*(256-I)+X1*I)>>8
        Color[X][Y]=mri[N][I][J]

// (X,Y) : 2D coordinate in the drawing plane
   mri[N][I][J] : (I,J) point in the Nth MRI
```

2.3 The simulated surgery

After constructing the 3D medical image, users may want to cut a certain portion of the image to see what is inside. That is, to hold a simulated surgery. In our system, users can achieve this goal easily. First, use a mouse to create several control points on the image. The polygon enclosed by these points is the section that users want to cut. Secondly, users must specify the depth to be cut. With this depth, the cut volume will be specified, and all the data in this volume must be removed.

To check that whether a point falls into the cut volume, the following method is used: for every side of the polygon, calculate its linear equation with direction. If a point falls into the polygon, the values which are evaluated by the linear equations will be all positive or all negative (depends on the direction of linear equations). If the point is out of the polygon, some of the values will be positive and some will be negative.

After finishing the depth test, the cut volume is decided. The cutting operation is very easy. Just set the color(opacity) of points inside the cut volume to zero, and then display the volume again. Users can then see the result image after the cutting operation. Users can also rotate or scale up the result image to see more clearly the inside structure of the human body, thus achieving the goal of simulated surgery.



$F(x,y)=(y1-y0)x+(x0-x1)y+(x1*y0-x0*y1)=0$

F(x,y) : linear equation with

3. Conclusion

Using the speed-up methods mentioned above, we can fast construct and rotate the MRI data. The following table lists the construction time and the rotation time on different computers. The data being used are 55 256x256 MRI bitmap images.

	Construction Time (seconds)	Rotation Time (seconds)
Pentium 166 32 M RAM	**2.483**	**2.123**
Pentium II 233 64 M RAM	**1.862**	**1.482**
Pentium II 300 64 M RAM	**1.452**	**1.142**

Acknowledgement

Thanks to professor Cheng-Kung Cheng, the chairman of Institute of Biomedical Engineering in National Yang-Ming University, for providing the visible human body MRI data of Taiwan.

Reference

[1] Philippe Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. Proceedings of SIGGRAPH ’94. Computer Graphics, 451-458.

[2] Danskin, John and Pat Hanrahan. Fast algorithms for volume ray tracing. 1992 Workshop on Volume Visualization, 91-98, Boston, MA, October 1992.

[3] Laur, David and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. Proceedings of SIGGRAPH ’91. Computer Graphics, 25(4): 285-288, July 1991.

[4] Jayaram K. Udupa and Dewey Odhner. Fast Visualization, Manipulation, and Analysis of Binary Volumetric Objects. IEEE Computer Graphics & Applications, 1991, 53-62.

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

[6] Marc Levoy. Display of surfaces from volume data. IEEE Computer Graphics & Applications, May 1988, 29-37.

[7] Pei-Wen Liu, Lih-Shyang Chen, Su-Chou Chen, Jong-Ping Chen, Fang-Yi Lin, and Shy-Shang Hwang. Distributed Computing: New Power for Scientific Visualization. IEEE Computer Graphics and Applications, Vol. 16, No. 3, May 1996, 42-51.

[8] Schröder, Peter and Gordon Stoll. Data parallel volume rendering as line drawing. Proceedings of the 1992 Workshop on Volume Visualization, 25-32, Boston, October 1992.

[9] Zuiderveld, Karel J., Anton H.J. Koning, and Max A. Viergever. Acceleration of ray-casting using 3D distance transforms. Proceedings of Visualization in Biomedical Computing 1992, 324-335, Chapel Hill, North Carolina, October 1992.

[10] Herman, G., Liu, H. 3D Display of Human Organs from Computer Tomograms. Computer Graphics and Image Processing, Vol. 9, No.1, 1-21, January 1991.

[11] Sakas, G. Interactive Volume Rendering of Large Fields. The Visual Computer, Vol. 9, No. 8, 425-438, August 1993.

[12] Stata, A., Chen, D., Tector, C., Brandl, A., Chen, H., Ohbuchi, R., Bajura, M., Fuchs, A. Case Study: Observing a Volume Rendered Fetus within a Pregnant Patient. Proceedings of Visualization ’94, 364-368, 1994.

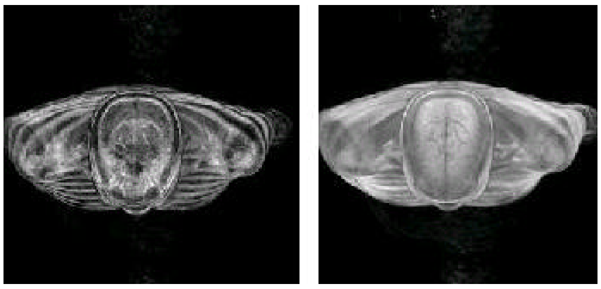
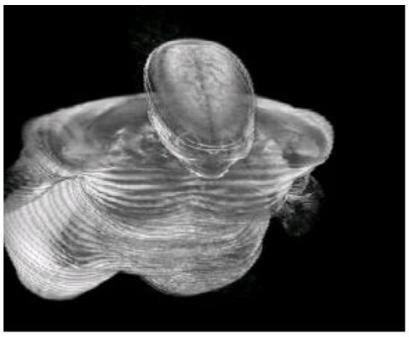
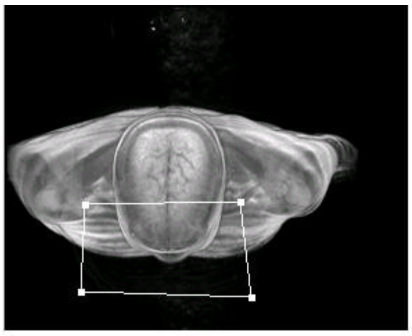


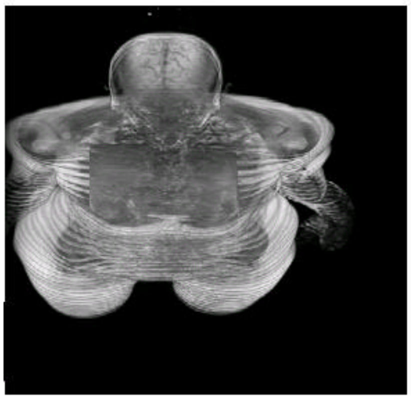
Figure 1: Ray Tracing with the original method (left) and with the proposed method (right)



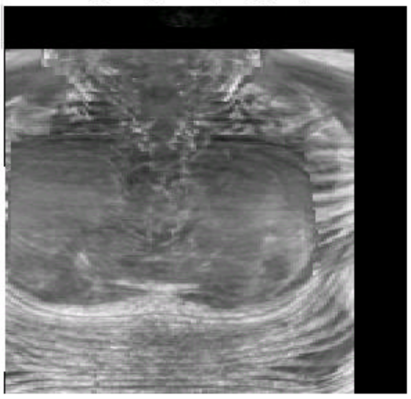
3D image after rotation



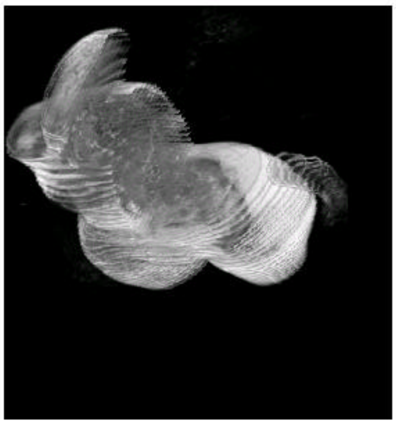
user interface for cutting
(the cutting section is enclosed by the high-lighted polygon)



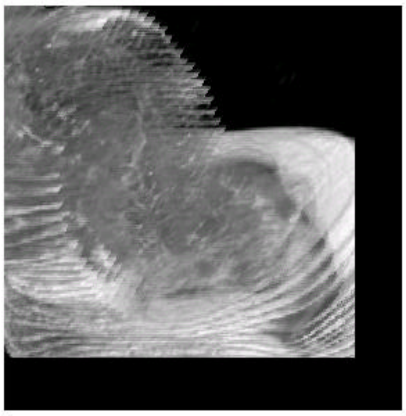
3D image after cutting and rotation (I)



3D image after scaling up the image to the left



3D image after cutting and rotation (II)



3D image after scaling up the image to the left