Efficient Volume Rendering by IsoRegion Leaping Acceleration

Ping-Fu Fung
pffung@cse.cuhk.edu.hk

Pheng-Ann Heng
pheng@cse.cuhk.edu.hk

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong

Abstract

This paper introduces a new acceleration technique for ray casting type volume rendering called IsoRegion Leaping. A factorization of the sample composition equation leads to our new algorithm which is based on homogeneity inside a volume. We create an IsoRegion data structure to help identifying homogeneous voxel cubes, which is independent of the viewing parameters and shading conditions, thus can be pre-processed. Accumulated colors and transparencies of homogeneous ray segments are pre-computed into a look-up table according to different voxel values and ray segment lengths. Attributes of ray segments piercing through IsoRegion blocks are looked-up from the table. Therefore, unnecessary sample compositions within IsoRegions are saved. This voxel leaping technique is experimentally proved to be efficient even for real-life medical volume data. A speedup of 2 to 3 times is measured while preserving the image quality. Images generated by our algorithm and the ones from the original algorithm without IsoRegion Leaping acceleration are identical, both theoretically and experimentally.

Keywords: volume visualization, ray casting, IsoRegion

1 Introduction

Volume rendering is playing an important role in computer visualization for displaying 3D volumetric data. It is a technique for projecting a 3D volume onto a 2D image plane. This process involves considerable amount of computation effort. This paper introduces a new acceleration technique for ray casting type volume rendering by exploiting homogeneity inside the data volume and avoiding unnecessary sample compositions. It is called IsoRegion Leaping.

Various acceleration techniques for volume rendering have been introduced so far. There are algorithms exploiting temporal coherence for a sequence of renderings such as the algorithm proposed by Law and Yagel [11]. The shear-warp factorization method by Lacroute and Levoy [10] [9] defines a specialized traversal scheme to exploit image space and object space coherences. Adelson *et al.* [1] [3] [2], He and Kaufman [7] studied stereoscopic coherence to help generating stereoscopic pairs more efficiently. Yagel and Kaufman [15] introduces the template-based ray traversal scheme which is the basis of a class of techniques employing ray coherence. Other acceleration methods make use of hardware assistance such as 3D texture mapping by Fang [6] and Kulick [8] as well as parallel/distributed rendering by Lacroute [9] and Ma [13].

Observe that most of the mentioned algorithms rely on various kinds of coherences: temporal, object-space, image-space, stereoscopic and (ray) sample-space. This enlightens us to develop an acceleration technique which exploits the coherence (homogeneity) lies inside the volume (object-space).

In the following sections, the basic compositing projection procedure is studied and a factorization of the sample composition equation is derived. Accompanied with this result, we show that the pre-multiplication of the color and opacity transfer functions, which is a common acceleration technique, may introduce some errors. With the factorization result, we develop the idea of IsoRegion Leaping acceleration. This technique pre-computes the accumulated colors and transparencies of ray segments within homogeneous voxel regions (IsoRegions). With this new technique, the image quality is not sacrificed while achieving a speedup of 2 to 3 times. This is demonstrated in our experiment results follow.

2 Compositing Projection

In compositing projection, semi-transparency is simulated by assigning opacity (or transparency) attributes to voxels. The **over** operator combines the colors and opacities of the samples pierced by the ray. This process is known as sample composition:

$$\begin{aligned} R_{final} &= C_1\alpha_1 + (1-\alpha_1)C_2 \\ A_{final} &= 1-(1-\alpha_1)(1-\alpha_2) \end{aligned}$$

where R_{final} is the color after composition, A_{final} is the accumulated opacity, C_i and α_i denote the color and opacity of the i^{th} sample. Note that the sample with smaller index locates at the side of the ray source (observer).

To make the subsequent discussion more concise, we also introduce the transparency term β and accumulated transparency B:

$$\begin{aligned} \beta_i &= 1-\alpha_i \\ R_{final} &= C_1\alpha_1 + \beta_1 C_2 \\ B_{final} &= 1 - A_{final} = \beta_1\beta_2 \end{aligned}$$

This formulation leads to a more efficient implementation by reducing the number of operations.

Eqn. 1 defines the pre-multiplication of the color and opacity values of the samples. Storing this pre-multiplied value is to improve performance [12] [5]. However, we should notice that this pre-multiplied value may not be always accurate to be used. In case of point sampling of ray samples at exact voxel grid positions, there is no problem since this pre-multiplied value will be the exact value we need. Other sampling circumstances using interpolations and/or filters applied directly on this pre-multiplied value will result in slightly different sample value. Here is an example of using linear interpolation of ratio $u : (1-u)$ to obtain a sample between two voxels p and q:

$$\begin{aligned} M_i &= C_i\alpha_i \qquad (1) \\ C_{sample} &= uC_p + (1-u)C_q \\ \alpha_{sample} &= u\alpha_p + (1-u)\alpha_q \\ M_{sample} &= C_{sample}\alpha_{sample} \\ &= [uC_p + (1-u)C_q][u\alpha_p + (1-u)\alpha_q] \\ M'_{sample} &= uM_p + (1-u)M_q \\ &= uC_p\alpha_p + (1-u)C_q\alpha_q \end{aligned}$$

It is apparent that M'_{sample} is not an accurate result, but only an approximation to M_{sample}. Nevertheless, due to efficiency concern and the fact that the error introduced is often not noticeable, some existing rendering algorithms and systems do store this pre-multiplied value [12] [5]. Moreover, suitable adjustments to C_i's according to the underlying re-sampling scheme may make this approximation fit better.

To composite a train of n samples along a ray, we assume each of them is associated with a tuple consisting of color C_i and opacity α_i:

$$(C_1,\alpha_1), (C_2,\alpha_2), ..., (C_n,\alpha_n)$$

According to the transparency formula (recursive application of the over operator) [12] [13], the accumulated color of the ray, R_{final}, is:

$$\begin{aligned} R_{final} &= C_1\alpha_1 + \beta_1(C_2\alpha_2 + \beta_2(C_3\alpha_3 + \beta_3(...))) \\ &= C_1\alpha_1 + \beta_1 C_2\alpha_2 + \beta_1\beta_2 C_3\alpha_3 + ... \\ &= \sum_{i=1}^{n}\left[C_i\alpha_i\prod_{j=1}^{i-1}\beta_j\right] \qquad (2) \end{aligned}$$

To evaluate this expression, there are two well-known strategies: back-to-front ordering and front-to-back ordering. For back-to-front ordering, this evaluation sequence is observed:

$$\begin{aligned} R_n &= C_n\alpha_n \\ R_i &= C_i\alpha_i + \beta_i R_{i+1} \\ R_{final} &= R_1 \end{aligned}$$

This evaluation sequence starts from the inner terms and propagates the results outward in $O(n)$. On the other hand, the front-to-back ordering performs the task in this way:

$$\begin{aligned} (R_1, B_1) &= (C_1\alpha_1, \beta_1) \\ (R_{i+1}, B_{i+1}) &= (R_i + B_i C_{i+1}\alpha_{i+1}, B_i\beta_{i+1}) \\ R_{final} &= R_n \end{aligned}$$

This set of equations effectively breaks down the parentheses and evaluates the sum terms one by one in $O(n)$. B_i's are used to boost the performance by keeping track of the growing accumulated transparency product term. The front-to-back ordering gains more attention because it allows early ray termination, which is an efficient acceleration technique for compositing projection proposed by Levoy [12].

After factorizing **Eqn. 2**, we get:

$$\begin{aligned} R_{final} &= \sum_{i=1}^{m}\left[C_i\alpha_i\prod_{j=1}^{j=i-1}\beta_j\right] \\ &+ \left[\prod_{k=1}^{m}\beta_k\right]\sum_{i=m+1}^{n}\left[C_i\alpha_i\prod_{j=m+1}^{j=i-1}\beta_j\right] \qquad (3) \end{aligned}$$

$$= \quad C_1^* + B_1^* C_2^*$$
$$\text{for any } m \in [1, n]$$

For any m lying within $[1, n]$, this factorization result is valid. The significance of this factorization is that it provides the foundation of sample composition by parts. Following the front-to-back ordering, we divide the list of samples along the ray into segments. By compositing the samples within each segment in front-to-back order, we obtain:

$$(C_1^*, B_1^*), (C_2^*, B_2^*), ..., (C_t^*, B_t^*)$$

where t is the number of segments, C_i^* and B_i^* are the accumulated colors and transparencies of the segments respectively. By recursive application of the factorization result, we have:

$$R_{final} \quad = \quad \sum_{i=1}^{t} \left[C_i^* \prod_{j=1}^{j=i-1} B_j^* \right] \tag{4}$$

Ma has developed a special form of this factorization for binary case [13]. **Eqn. 3** is a more generalized result. The IsoRegion acceleration scheme, which will be discussed in the following section, is mainly based on the result of **Eqn. 4**.

3 IsoRegion Leaping Acceleration

Space leaping is a well-known acceleration technique for ray casting type volume rendering by skipping empty space [16]. There are several variants such as Proximity-clouds by Cohen and Shefer [4], Ray Acceleration by Distance Coding by Zuiderveld *et al.* [17] and Macro-regions of Empty Space by Šrámek [14]. All these space leaping techniques accelerate the ray casting process by avoiding unnecessary, time-consuming empty space traversal. To apply these algorithms, we have to first classify the empty space (background voxels) by thresholding, segmentation and/or statistical methods. This classification process often introduces inaccuracy and requires human intervention.

Unlike the mentioned algorithms which only skip empty space, voxel regions with identical voxel values are also eligible to be leaped under the concept of region leaping, leading to further speedup. Such voxel regions are called IsoRegions. With IsoRegions, we need not to classify the empty space since all kind of voxels are considered and may be leaped. Ray samples falling in different IsoRegions are composed into ray segments. These ray segment results are accumulated along the ray in front-to-back order using the factorization result of **Eqn. 4**. Because the ray samples within an IsoRegion take the same voxel value, pre-computation of the ray segment colors and transparencies is possible. This is the main source of improvement.

Since IsoRegion is the heart of this new algorithm, firstly we discuss how the IsoRegions are defined and constructed. Then, we introduce the pre-computed ray segment accumulated color and transparency table, followed by the ray traversal scheme.

3.1 IsoRegion Definition

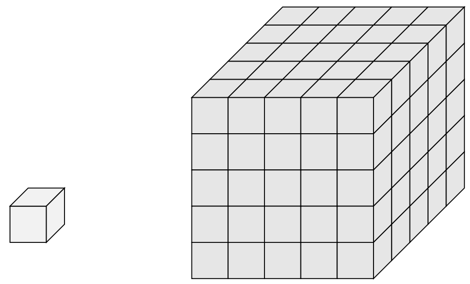


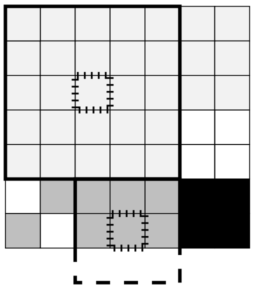
Figure 1: Definition of an IsoRegion

There are two example IsoRegions in **Fig. 1**. An IsoRegion is an odd dimension cubic region of voxels having identical voxel value. Each IsoRegion consists of $(2d+1)^3$ voxels for some natural number d. Hence, the smallest IsoRegion contains only one voxel which is a special case with $d = 0$. Given an IsoRegion of dimension d, centered at (x, y, z), the following holds:

$$V(x+i, y+j, z+k) = V(x, y, z), \forall i, j, k \in [-d, d]$$

For i, j and k ranging in $[-1, 1]$, there is a set of $3^3 - 1 = 26$ immediate neighbours (N_{26}). Such close neighbourhood relationship is said to be 26-connected, whereas 8-connected (3^2-1) in 2D case. This definition ensures the voxel located at the center of the IsoRegion is surrounded by neighbours of the same voxel value upto d 26-connected steps. Such a 26-connected neighbourhood distance is also known as chess board distance [14]. From the center of an IsoRegion, we can leap over d voxel grid distance in all directions and be sure that we still encounter voxels taking identical voxel value.

To store the IsoRegion information, only the dimension parameter d of the IsoRegions are recorded at the center of the IsoRegions. **Fig. 2** shows an example for 2D case. The IsoRegions are constructed disregarding what the voxel values are (there are four kinds of voxels in this illustration). Notice how the IsoRegions extent



Volume voxels

4	3	2	2	2	2	2
3	3	2	1	1	1	1
2	2	**2**	1	0	0	0
1	1	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	**1**	0	0	1

IsoRegion information

Figure 2: Example of a 2D IsoRegion data structure

along the volume boundaries (shown in dotted line). For such cases, the boundaries are considered as extending homogeneously.

In the development of better space leaping techniques, there are researchers investigating how to give a tighter bounding region for empty space. Distance functions other than chess board distance to define a region have been proposed such as city block distance, Euclidean distance and chamfer distance [14]. With a looser bound, only conservative estimation of the IsoRegions is possible and results in generally smaller IsoRegions, however, with less computation effort. On the other hand, a tighter bound leads to larger IsoRegions, hence better voxel leaping speedup, but it takes more computation time in extracting the IsoRegion information. When defining our IsoRegion, these results are referenced and finally we choose chess board distance. Chess board distance is a compromise with respect to both aspects. It involves simple computation while providing a good estimate of the bound of an IsoRegion.

3.2 IsoRegion Construction

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

Algorithm 1 CreateIsoRegion

▷ *Initial step:*
for all voxel positions (x, y, z)
  if $V(x+i, y+j, z+k) = V(x, y, z)$
    $\forall i, j, k \in [-1, 1]$
  then
    IsoRegion$(x, y, z) \leftarrow 1$
  else
    IsoRegion$(x, y, z) \leftarrow 0$
▷ *Iteration step:*
$d \leftarrow 1$
repeat
  for all voxel positions (x, y, z)
    if IsoRegion$(\vec{w}) \geq d$
      $\forall \vec{w} \in N_D(x, y, z)$
    then
      IsoRegion$(x, y, z) \leftarrow$ IsoRegion(x, y, z) + 1
  $d \leftarrow d + 1$
until no further IsoRegion found

Although this processing step takes time, we only need to perform this pre-processing once for each volume data set and the result can be stored accompanied with the volume. This piece of valuable information is independent of the viewing parameters and shading parameters, thus we can always reuse it. Also notice that this pre-processed data structure consumes memory and takes a size comparable to the volume data set. However, present workstations are often equiped with huge amount of memory and current hardware capability is enough to cope with such size. Furthermore, this piece of information may be compressed, for example, using less number of bits to store the dimension parameters under the observation that very large step sizes (say, over 50) are rarely encountered, especially in real life captured data sets like medical MRI volume data sets. Storing a conservative estimation of these rarely occurred step sizes will not hurt but keep the memory consumption fit. For example, we can impose a maximum IsoRegion dimension of 15 and use 4 bits for each cell which results in a 50% reduction in memory consumption compared with 8-bit cells.

3.3 IsoRegion Step Table

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

$$\text{IRST} : (v, d) \mapsto (C^*, B^*)$$

where v and d are the voxel value ($[0, 255]$) and step value (dimension parameter, $[1, \text{max step}]$) of an IsoRegion respectively, C^* and B^* stand for the accumulated color and transparency of the resulting ray segment. The calculation of the IRST is based on **Eqn. 2**. Since all the voxels along the homogeneous segment take the same value, the values for C_i's are identical. Similar for α_i's and β_i's:

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

thus after simplification, we have:

$$\text{IRST}(v, d) \quad = \quad (C^*, B^*) \quad = \quad (C(1 - \beta^d), \beta^d)$$

With the IRST, we can retrieve the accumulated color and transparency values of a ray segment given the voxel value and step value. Notice that the sample distance along the ray is assumed to be the same as the voxel grid distance. In case the sample distance is not defined according to this assumption, we have to make the following adjustment to the step value:

$$\text{new step value} = \frac{d}{d_s}$$

where d_s is the sample distance along the ray.

3.4 Ray Traversal Scheme

The ray traversal scheme adopted is pretty much the same as the basic ray casting algorithm except the shading and sample composition parts. Our ray casting algorithm is basically a ray-driven (image-order) one with parallel compositing projection. We assume the volume data set is sampled uniformly and homogeneously along the three principle directions (i.e. cubic voxels). For simplicity, integral voxel grid re-sampling is applied. The caster shoots ray towards each pixel on the image plane from the view point. If the ray misses the volume, the corresponding pixel is shaded black. Otherwise, it pierces the volume and a train of samples is taken along its track. Ordinary ray casters take samples at evenly spaced positions along the ray. However, our algorithm takes samples at unevenly spaced positions depending on the size of the leaps over the IsoRegions encountered.

Fig. 3 illustrates how our ray traversal scheme works. Ray A firstly encounters an IsoRegion with dimension 3. An IRST look-up is performed (white dot)

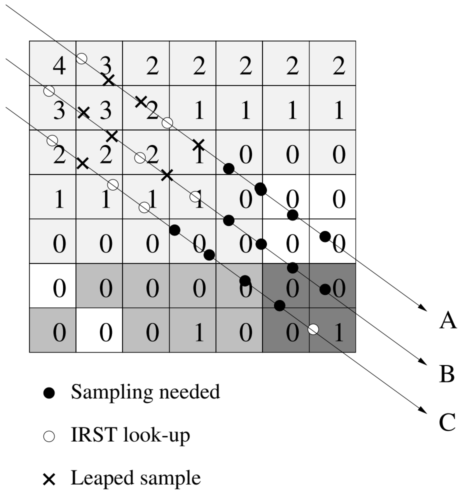


Figure 3: IsoRegion Leaping: unevenly spaced samples depending on IsoRegion dimension (leap value)

to retrieve the accumulated color and transparency of this ray segment (containing 3 samples). Therefore, 2 samples (cross marks) are leaped before taking the next sample. The second sample is taken in another IsoRegion of dimension 2 and this time it saves 2 samplings by IRST look-up. IsoRegions of dimension 0 indicates that no homogeneity coherence can be exploited and single sample traversal should be enforced (black dots). There will be no speedup in this case. Due to the homogeneity of an IsoRegion extends from the center to all the directions, rays passing through the IsoRegion center from any direction must generate a homogeneous ray segment. Therefore, this acceleration technique is independent of the viewing direction. As long as an IsoRegion with non-zero dimension is encountered, there is an acceleration leap.

Algorithm 2 SampleAndComposite(r_e, $\vec{r_d}$)

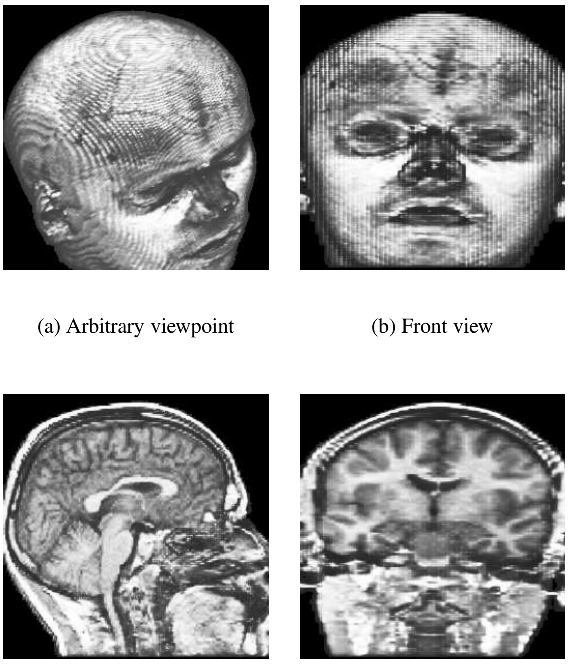
▷ *r_e is the coordinates of the entrance point of the ray to the volume*
▷ *$\vec{r_d}$ is the direction vector of the ray, with length equals to the sample distance*

```
C ← 0
B ← 1
while r_e ∈ volume V
    ▷ Sampling and shading step:
    NN(r_e) ← the nearest integral neighbour of r_e
    ▷ get the step value from the IsoRegion
    d ← IsoRegion(NN(r_e))
    if d = 0
    then
        ▷ no leaping, find sample color and transparency
        (C*, B*) ← Sample(r_e)
        ▷ Advance to next sample location
```

9 $\quad\quad r_e \leftarrow r_e + \vec{r_d}$
10 $\quad$ **else**
$\quad\quad \triangleright$ *get the voxel value from the volume*
11 $\quad\quad v \leftarrow V(NN(r_e))$
$\quad\quad \triangleright$ *IsoRegion ray segment look-up*
12 $\quad\quad (C^*, B^*) \leftarrow \text{IRST}(v, d)$
$\quad\quad \triangleright$ *Advance to next sample location*
13 $\quad\quad r_e \leftarrow r_e + \vec{r_d} \times d$
$\quad \triangleright$ *Compositing step:*
14 $\quad C \leftarrow C + B \times C^*$
15 $\quad B \leftarrow B \times B^*$
16 **end**

Line 8 indicates various re-sampling kernels may be applied. For example, integral voxel grid sampling and tri-linear interpolator. Incorporation of the early ray termination acceleration technique is also very simple. We just need to add one more checking condition for the accumulated transparency on line 3.

The resulting image is identical to the image generated by the ray casting algorithm without IsoRegion acceleration. This acceleration technique preserves the quality of the image due to the identity (the factorization result we derived) in **Eqn. 4**. It boosts the ray traversal efficiency by saving unnecessary sampling and compositions within IsoRegions, without affecting the final composited color of the rays.



(a) Arbitrary viewpoint (b) Front view

(c) Side view cut (d) Front view cut

Figure 4: Four renderings of a CT human head

4 Experiment Result

Fig	CPU time taken (sec)		Leap	Speed
	Original	IsoRegion	count	up
a	9.37	5.49	4157000	1.71
b	5.79	1.89	4700045	3.06
c	3.54	1.85	1692549	1.91
d	2.24	1.28	1399315	1.75

Table 1: Comparison of the original simple ray-casting algorithm and the IsoRegion Leaping accelerated one in terms of the rendering time and saving in sample composition for the renderings in **Fig. 4**

Fig. 4 shows four renderings of a human head with various rendering parameters. This medical data set was obtained through Computerized Tomography (CT) and is available from public domain. The data set is of dimensions $256 \times 256 \times 256$ with cubic voxels of 8-bit precision. The renderings were done on a Sun Ultra 1/170 with 512 MB main memory running Solaris 2.5.1. All the images are of size 256×256. **Table 1** summarizes the timing results of the original simple ray-casting algorithm and the one accelerated by the IsoRegion Leaping technique. The column leap count indicates how many sample compositions are saved by using the new technique. The recorded speedup is about a factor of 2 to 3 depending on the rendering conditions.

In this experiment, the images generated by the accelerated algorithm are identical to those generated by the non-accelerated one. This is proved theoretically from the factorization identity and experimentally. The re-sampling kernel applied is nearest integral neighbour voxel. We have to emphasize that the new technique is independent of the underlying re-sampling/filtering method. In case of using more sophisticated re-sampling method, such as tri-linear interpolation, our algorithm will even score a higher time saving in magnitude because the saving per sample is greater.

Notice that the actual gain depends on the data set and the rendering parameters. In case of synthetic volume data and data sets with lots of homogeneous regions, potentially greater number of large IsoRegions can be constructed resulting in greater leaping and speedup. For viewing directions encountering more homogeneous regions, better timing results are expected.

To apply the new acceleration technique, we have to perform some pre-processings beforehand. Firstly, the IsoRegion information has to be extracted and stored.

For the CT human head data set shown above, this step took 173 second. This step should be done once for each data set and is independent of the rendering parameters and the underlying rendering kernel used. The IsoRegion information is stored accompanied with the original volume data set. In our case, the CT human head data set takes 16 MB while the IsoRegion information takes 8 MB (4-bit cell). The IsoRegion information may be further compressed. Secondly, the IRST should be prepared. As long as the color and transparency transfer functions are fixed, we only need to do this pre-processing step once. It took 0.42 second in our experiment. If we are to do a sequence of renderings with the same transfer functions, such as the case of a fly through, the IRST can be reused for every renderings.

5 Future Work and Discussion

The IsoRegion Leaping Acceleration technique is readily applicable to perspective ray casters. It is because the IsoRegion is defined as a cubic bounding region, thus rays of any directions can be handled correctly and the leapings are always valid. Moreover, the rays are independent of each other so that both parallel and perspective viewing are accelerated. Further investigation and experiments are needed to prove this.

We are also working on accelerating other kinds of projection type using our IsoRegion technique. For instance, the X-ray type (maximum) projection will be our next target. Rather than performing sample compositions, the maximum densities are collected along the rays. This will be an interested topic of our future research.

It seems that our current approach only works for ray casting without lighting. However, if parallel viewing under directional light source is involved, our IsoRegion acceleration technique may still apply. Under such circumstances, the IRST has to be re-calculated for different viewing and lighting angles, not to be reused throughout a sequence of renderings. The rationale behind is due to the fact that the voxels inside an IsoRegion is homogeneous and thus the normals. Therefore, the shading effect for a homogeneous ray segment can be pre-calculated. This is a proposed approach and further research is required.

6 Conclusion

To summarize, we developed a new volume rendering acceleration technique taking the advantage of homogeneity inside a volume data set. The IsoRegion Leaping acceleration technique traverses homogeneous voxel blocks, no matter what kind of voxels they are. It improves over existing space leaping techniques in the sense that no background (empty) voxel classification is required and all sorts of voxels are eligible to be leaped. This technique does not sacrifice the image quality for speed as proved by our factorization result and experiment. It just simply avoids redundant sample compositions.

Experimental result with parallel projection on medical data set shows that this new acceleration technique boosts the speed of the simple ray casting algorithm by a factor of 2 to 3. This technique is readily applied to perspective viewing. Moreover, early ray termination technique may also be incorporated.

7 Acknowledgement

The authors would like to thank for the many constructive comments of the reviewers. The open question on light shading is suggested by one of the reviewers. Compliments also go to Helen Ng and Elton Tsang for proof-reading the drafts.

This work is supported by RGC Earmarked Grant CUHK4162/97E of Hong Kong.

References

[1] Stephen J. Adelson, Jeffrey B. Bentley, In Seok Chong, Larry F. Hodges, and Joseph Winograd. Simultaneous Generation of Stereoscopic Views. *Computer Graphics Forum*, 10:3–10, 1991.

[2] Stephen J. Adelson and Charles D. Hansen. Fast Stereoscopic Images with Ray-Traced Volume Rendering. *ACM/IEEE 1994 Symposium on Volume Visualization*, pages 3–9, 10 1994.

[3] Stephen J. Adelson and Larry F. Hodges. Stereoscopic Ray-Tracing. *SIGGRAPH 93 Course 16 Notes*, pages 4–24–53, 1993.

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

[5] John Danskin and Pat Hanrahan. Fast Algorithms for Volume Ray Tracing. *Proceedings 1992 Workshop on Volume Rendering*, pages 91–98, 1992.

[6] Shiaofen Fang, Rajagopalan Srinivasan, Su Huang, and Raghu Raghavan. Deformable Volume Rendering by 3D Texture Mapping and Octree Encoding. *IEEE Visualization '96 Conference*, pages 73–80, 1996.

[7] Taosong He and Arie Kaufman. Fast Stereo Volume Rendering. *IEEE Visualization '96 Conference*, pages 49–56, 1996.

[8] Todd Kulick. Building an OpenGL Volume Renderer. *SIGGRAPH Course Notes 1996*, pages 1–8, 1996.

[9] Philippe Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. *Proceedings of the 1995 Parallel Rendering Symposium*, pages 15–22, 1995.

[10] Philippe Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *SIGGRAPH '94, In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH*, pages 451–458, 1994.

[11] Asish Law and Roni Yagel. Exploiting Spatial, Ray, and Frame Coherency for Efficient Parallel Volume Rendering. *Proceedings of GraphiCon '96*, 2:93–101, 6 1996.

[12] Marc Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9(3):245–261, 7 1990.

[13] Kwan-Liu Ma and James S. Painter. Parallel Volume Visualization on Workstations. *Computer and Graphics*, 17(1):31–37, 1993.

[14] Miloš Šrámek. Fast ray-tracing of rectilinear volume data. *Virtual Environments and Scientific Visualization '96, Proceedings of the Eurographics Workshops*, pages 201–210, 1996.

[15] Roni Yagel and Arie Kaufman. Template-Based Volume Viewing. *Proceedings of EuroGraphics '92*, pages 153–167, 9 1992.

[16] Roni Yagel and Zhouhong Shi. Accelerating Volume Animation by Space-Leaping. *Proceeding of Visualization '93*, pages 62–69, 1993.

[17] Karel J. Zuiderveld, Anton H. J. Koning, and Max A. Viergever. Acceleration of ray-casting using 3D distance transforms. *Visualization in Biomedical Computing II, Proc. SPIE 1808*, pages 324–335, 1992.