Accelerating Volume Visualization by Exploiting Temporal Coherence

Ilmi Yoon, Joe Demeres, Taeyong Kim, Ulrich Neumann
Computer Science Department
University of Southern California
Los Angeles, CA 90008
iyoon@graphics.usc.edu

Abstract

*In this paper we present two methods for accelerating surface visualizations of volume data, the **image cache** and the **isomap**. The image cache allows us to take advantage of temporal coherence during viewpoint changes. The image cache, which maintains intermediate values used to render the previous image, can be queried for visibility and shading information. While changing the isosurface threshold values, an isomap is generated for the current camera position. This new structure allows surface threshold variations to be rendered interactively. These methods perform well with large data sets because they reduce the rendering time dependence on data size.*

1. Introduction

Volume visualization techniques enable the comprehension of 3D sampled scalar or vector fields. These techniques are mainly categorized into two types. Surface extraction methods generate polygon meshes. These techniques include binarization [Her91], marching cubes [Lor87], contour connecting [Fuc77]. These methods take advantage of fast rendering during surface visualization, however the surface meshes are generally not created in real time. The other visualization approach is direct volume rendering which projects the entire volume data into an image without generating intermediate geometric representations. Opacity mapping functions, or classifiers, are used to assign color and opacity to each voxel based on its value. These techniques include ray casting [levoy88], splating [Wes90], shear-warp factorization [Lac95].

Visualization becomes arduous as the volume data size increases. Atmospheric, geologic, simulation, and medical applications deal with enormous data sets. For example, the male "Visible Human" model [VHM] has 122 Mvoxels of MR data, 490 Mvoxels of CT data, and about 14 Gvoxels of photo-texture data. In most cases, the number of pixels in the image is much lower than the number of volume elements (voxels or polygons). We take advantage of this relationship through the methods we present here by reusing data from previous images to produce new ones.

Recent image-based rendering techniques also exploit the benefits of bounded image size. View interpolation [Che93] requires multiple initial images with correspondences, from which new views can be generated. View morphing [Sie96] takes advantage of some knowledge of the underlying 3D geometry to preserve shape during view interpolation and greater freedom of motion between images. These methods work for rigid static models, but interactive classification produces a changing volume model which our approachs handle.

Temporal coherence, information from a frame which can be used to accelerate to render next frame, is used in [Yag93], where each pixel's closest opaque voxel coordinate is stored in a C-buffer (coordinate buffer) for use in the next frame. The C-buffer coordinates are transformed to compute the start point for ray casting a new image. Since the C-buffer stores point samples, aliasing may be propagated from frame to frame. Also, no temporal coherence is possible in the case where the opacity mapping function is changed.

In this paper, we introduce two methods, the *image cache* and the *isomap*, for accelerating visualization of volume data, by exploiting temporal coherence. The image cache allows us to extract information from previously rendered frames. When a user stops manipulating the view point and begins interactively changing the isosurface threshold value, an isomap is generated for the current camera pose. This isomap structure accelerates the display of varying threshold surfaces in real time.

Section 2 begins with brief system overview. Section 3 introduces the image cache, and section 4 describes the isomap. In section 4 & 5, we discussed our results and future research.

2. System Overview

Our system is based on volume renderer, ray casting system but for more focusing on displaying isosurface, we used single surface value as a threshold instead of opacity mapping function. In this way, it is usually much faster than ray casting with opacity mapping function. It is slower than surface rendering but there is no need to preprocess to extract isosurface. In our system isosurface and polygons are taken care in same way to extract information from previous frame. But if not only isosurfaces but also non fully opaque area needs to be displayed, it can be more generalized by storing first non transparent voxels in im-

age cache. Our current system takes almost same time for view change and threshold value change, and we accelerate view change by using of image cache and threshold value change with isomap.

The outer loop of our renderer performs the same as a traditional ray caster, casting a view ray through each pixel which is then clipped to the volume's boundaries. Each view ray is then passed to the image cache's query method, which returns the information available in the image cache along that ray. The possible responses are *definite miss*, *definite hit* or *possible hit.* Definite miss indicates that the ray dose not intersect isosurface, and results in the current pixel being set to the background color. Definite hit indicates surface was hit, and the 3d position at which the intersection occurred can be computed from the distance returned, allowing the current pixel to be shaded without needing to perform any intersection tests. Possible hit indicates that the volume may have been hit, and provides a 3d world position at which to begin ray traversal. The traditional renderer is then invoked on the remainder of the ray to determine visibility and shading for the current pixel.

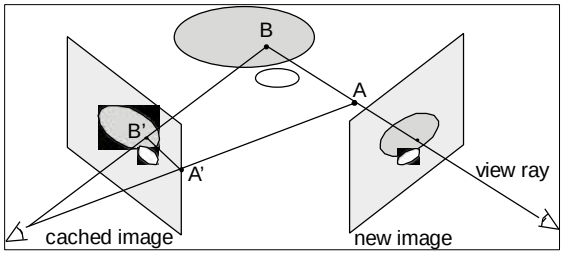


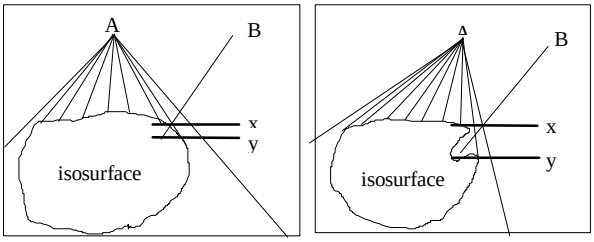
Fig. 1 - projection of ray AB to A'B in cache image

3. Image Cache

In the general case, the image cache holds some number of previously rendered images. Each of these images' pixels contain at least depth information, indicating where the visible surface within that pixel lies. Color and normal can also be stored and reused by interpolation. In our implementation, the image cache holds exactly one image, and so for simplicity's sake, when we refer to the image cache, we are also referring to the image stored therein.

Information along a ray is extracted from the image cache by looping across the pixels that lie along the projection of the ray until the ray's depth exceeds the pixel's depth. First, the ray is projected onto the image plane and clipped to the image's extents, if need be. Pixels whose areas the ray falls within are traversed, along the ray's direction, with the ray's computed depth at each pixel being tested against the pixel's depth. If the ray's depth exceeds the pixel's depth, then we assert that the ray has passed behind the pixel's surface, which is not necessarily to say that the ray intersected the surface. It may have merely passed behind the surface. If the ray passed in front of the surface in the last pixel, then we assert that the surface was intersected, and return *definite hit*, along with the distance traversed along the ray. If, however, the surface wasn't visible in the previous pixel, then we return *possible hit*, along with the distance traversed along the ray, as the ray may or may not actually hit the volume. If the ray leaves the volume's boundary before leaving the image, then we can return *definite miss*, and lastly, if the ray leaves the image before leaving the volume's boundary, we must return *possible hit*, as the ray may intersect a portion of the volume which was previously offscreen.

Potential problem when using the image cache to extract visibility information lies in the assumption that two neighboring pixels with the same object lie near one another. Since volumes can have concavities, this is not necessarily the case. Fortunately, finding these concavities is a tractable problem. If the depths of the two neighboring pixels differ by more than one voxel distance, it is likely that this corresponds to a concavity, and so *possible hit* should be returned, along with the distance along the ray. Otherwise, the two pixels are on similar points on the surface, and points between them do not diverge greatly.



(a) definite hit (b) possible hit

Fig. 2 - concavity test - in (a) A is old camera (image cache), B is new camera. x and y (depths of the two neighboring pixels) is within 1 voxel distance (reuse color or reshading) - in (b) x and y are farther than 1 voxel distance (concavity, cast regular ray with given depth)

4. Volume Isomap

Changing surface value (threshold) of isosurface rendering is a traditionally slow operation. Even for the direct volume rendering, changing opacity mapping function is not fast enough to do it in real time or interactively. For example, shear-warp factorization algorithm provides very fast volume rendering for the view changes, but due to the run-length encoding, changing opacity map causes reconstruction of the run-length encoding.

Before starting those slow operations, volume isomap can be a great use even though each isomap is for a static camera. It displays surfaces of different values interactively in real time, not respect to the data size. This kind of functionality is more useful for some applications which have more focuses on how isosurfaces are constructed or

how surfaces moves when threshold values are changed. Building time of isomap is similar to the time taken to render one image of the volume with same view point, ray casting with a steadily increasing opacity mapping function without using early ray termination. Memory requirement is $C*n^2$ (n = image size, C = constant depending on data nature and user control). Not only that purpose, in our method, image generated from isomap can be used as image cache, too. Therefor, in sequences of changing viewpoints and then changing surface values can both utilize previously rendered image.

The isomap generates and stores a piecewise linear approximation of the change in volume values along each view ray for a given camera position. After eliminating all but the potentially visible portions of the approximating function, we are left with a function monotonically increasing in value. (Fig??) Either shading information or voxel position is also stored at each value point in the function where the function's slope is changed . This allows isosurfaces to be found very quickly by performing a binary search within the approximation function at each view ray, and then just interpolating stored color, or interpolating depth and reshading.

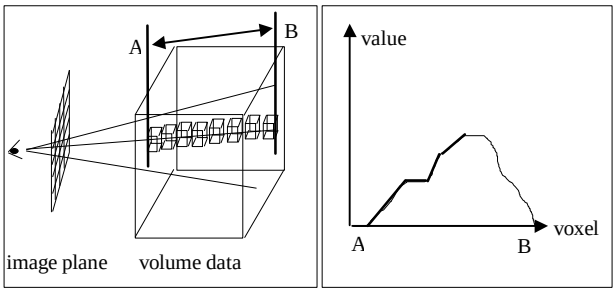


Fig. 3 - Isomap - in left, ray is casted from each pixel (A, B means starting and ending voxel) -in (b) curve shows change of voxel values through the casted ray and thick piecewise line is the approximation function.

Combining this with other strategies based upon knowledge of the previous surface value and the previous pixel's depth allows for impressive speedups. For example, threshold value increases can only shrink an isosurface, so cached image pixel-depths can be used as starting points for resampling the volume in search of the new threshold. We use color interpolation in our implementation, and can render surfaces in 1-1.5 seconds, for any of our data sets (engine data, brain, head and sand molecular data). The creation of the isomap itself takes between 1 and 1.5 times as long as creating a single image, which is easily worth the factor of 30 speedup we attain when changing threshold.

5. Results

We implemented our system on SGI Onix. For image cache, we tested two different cache replacement algorithms while generating animations: First is to keep updating image cache with the most recent image (fig 4) and second is to replace cache every 7 frames with a rendered image without using cache (fig 5). The first one gave better speedup, an there was no visible degradation over time from use of the image cache. Overall, images rendered with information from the image cache tended to be generated in one half to one third the time of those rendered without. Fig 8 above two shows images used as cache for both the brain (128x128x84) and 4 spheres, and engine (256x256x110) and below two shows images rendered using the image cache. Yellow pixels are only ones needed to be rerendered because of concavity test or new portion.

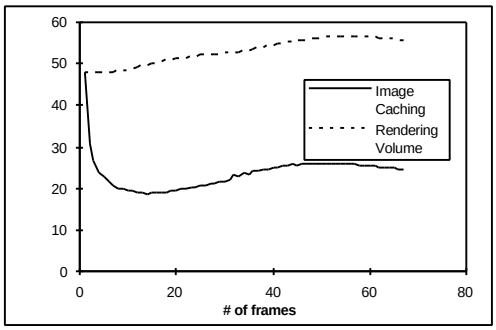


Fig. 5 - comparison of average times for images rendered using image cache vs. those rendered without. Cache is replaced with most recent image.

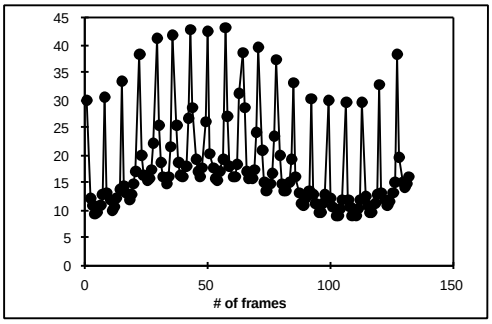


Fig. 6 - time taken to render images when one of seven images is chosen as the cached image (peak points) and the rest use their nearest cached image.

We enlarged the brain data set to twice its size in each dimension (128*128*84 --> 255*255*167), to compare similar data sets for a more robust comparison. The time for the larger data set was 75 seconds for rendering without image cache, and 20 seconds with. For comparison, the smaller data set was 50 seconds for rendering without image cache, and 15 seconds with.

Fig 7 shows the use of the isomap allowing scenes which were originally rendered in 35-40 seconds to be rendered at any surface value in 1-1.5 seconds after only 40-75 seconds preprocessing - not much more than the time to render a single image. It shows almost same results to 128*128*84 volume data and 255*255*167 volume data and data sets of different characteristics. Fig 9 shows images generated from isomap.

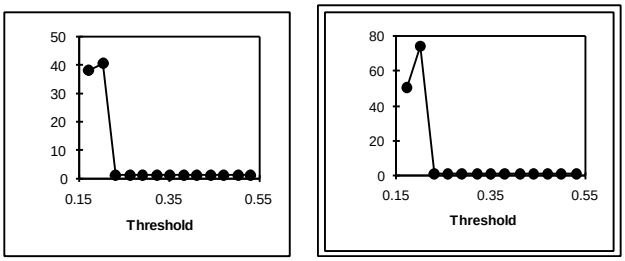


Fig. 7 - time to render an image without isomap(1st point), the isomap building(2nd point) and subsequent images with differing thresholds. Left is from the brain at 128x128x84 and right is 255x255x167.

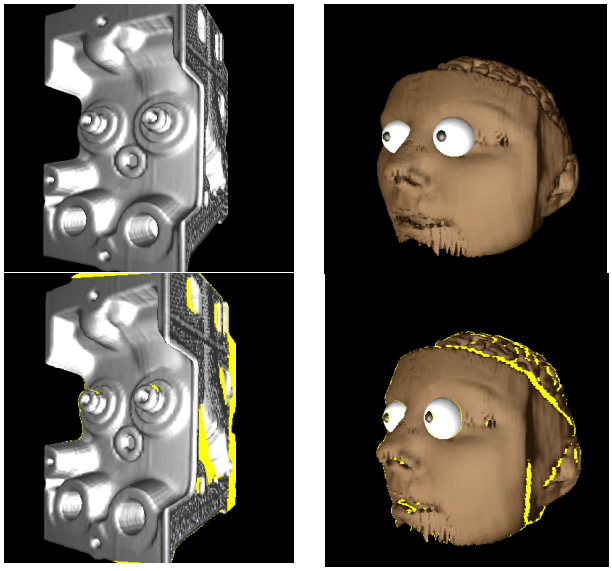


Fig. 8 - below two pictures are rendered from image cache, above two. Yellow means miss from the cache.

6. Future Research

In this paper, we presented two methods, image cache and isomap, to accelerate rendering sequencies. Using image cache showed factor of 2 to 4 speed up, without image degradation unless data has concavity whose width is less than one voxel distance, which seldomly happens. Isomap also showed great speed up in changing surface values without any visible image degradation. For future research, we plan to apply this system to irregular data sets which can gain large benefit by saving time through reusing information from previously rendered images. Also, object-orineted rendering method such as shear-warp factorization is considered to exploit tempolar coherence.

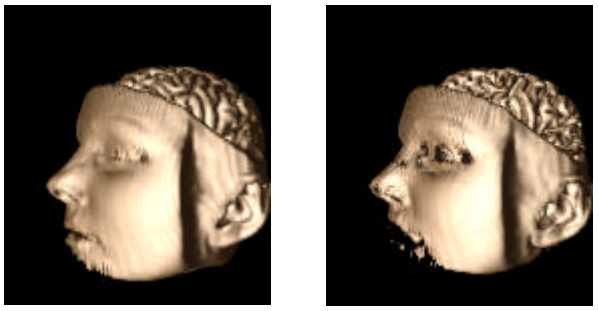


Fig. 9 - images generated from isomap. Threshold for left one is 0.16, and right one is 0.22.

7. References

[VHM] Visible Human Project web page http://www.nlm.nih.gov/research/visible/visible_human.

[Che93] S. Chen, L. Williams "View Interpolation for Image Synthesis," Proceedings of Siggraph 93 (Anaheim, CA), In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 279-288.

[Fuc77] H. Fuchs, Z. Kedem, S. Uselton, "Optimal Surface reconstruction from Planar Contours", Comm. of the ACM, Vol. 20, No. 10, pp. 693-702, 1977.

[Her91] G. Herman, H. Liu, "3D Display of human organs from Computer Tomograms", Computer Graphics and Image Processing, Vol. 9, No. 1, pp. 1-21, Jan 91.

[Lac95] P. Lacroute, M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform", Computer Graphics vol. 28, pp 451-458, Annual Conference Series 1994.

[Lev88] M. Levoy, "Display of Surfaces from Volume Data," IEEE Computer Graphics and Applications, Vol. 8, No. 3, May 1988, pp. 29-37.

[Lev90] M. Levoy, "Efficient Ray Tracing of Volume Data," ACM Transactions on Graphics, vol. 6, pp. 2-7, July 1990

[Lor87] W. E. Lorensen, H. E. Cline "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," Proceedings of Siggraph 87 (Anaheim, CA), In Computer Graphics Proceedings, Annual Conference Series, 1987, ACM SIGGRAPH, pp. 163-169.

[Sie96] S. Sietz, C. R. Dyer "View Morphing," Proceedings of Siggraph 96 (New Orleans, LA), In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 21-30.

[Wes90] L. Westover, "Footprint evaluation for volume rendering", In Computer Graphics Proceedings, Annual Conference Series, 1990, ACM SIGGRAPH, pp. 367-376.

[Yag93] R. Yagel, Z. Shi, "Accelerating Volume Animation by Space-Leaping", Proceedings Visualization 1993, IEEE, pp 62-69.