Fast Rendering of Foveated Volumes in Wavelet-based Representation

Hang Yu, Ee-Chien Chang, Zhiyong Huang, Zhijian Zheng
School of Computing, National University of Singapore
{yuhang,changec,huangzy,zhengzhi}@comp.nus.edu.sg

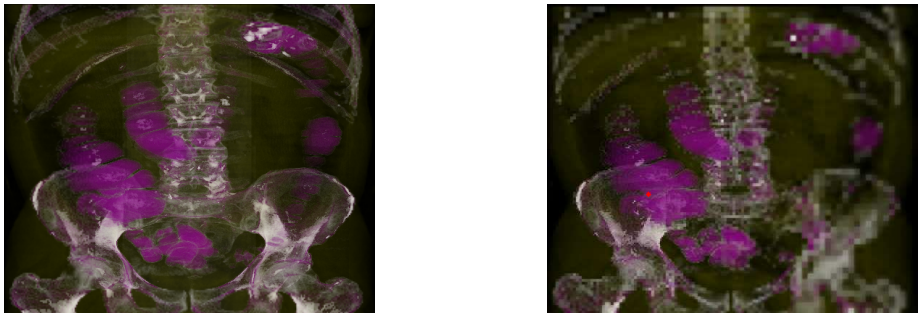
Abstract

A foveated volume can be viewed as a blending of multiple regions, each with a different level of resolution. It can be efficiently represented in the wavelet domain by retaining a small number of wavelet coefficients. We exploit the arrangement of those wavelet coefficients to achieve fast volume rendering. The running time is $O(n^2 + m)$ where n is the width of the rendered image, and m is the number of wavelet coefficients retained for the foveated volume. Our algorithm consists of two phases. The first phase is a fast reconstruction of the *super-voxels* from the wavelet coefficients, and the second phase renders the super-voxels by carefully tracking rays with different thickness in the super-voxels. Excluding the forward wavelet transformation on the original volume, no other preprocessing on the coefficients is required. Hence, it is possible to interactively modify different viewing parameters like the transfer functions. A potential application of our algorithm is in remote visualization of large volume data-sets.

1 Introduction

Volume visualization inherently consumes a huge amount of computing resources due to the large data size. Although there have been significant advances in volume rendering techniques and graphics hardware, real-time rendering of large volume data-set, for example, a data-set with 512x512x512 voxels, is still infeasible in current general purpose desktop PCs. The size of the data also poses technical challenge in remote visualization, where a viewer can interactively visualize a volume data-set stored in a remote server. In the context of remote visualization, there are two strategies of rendering, *render local* and *render remote* [4]. Under the first strategy, the whole volume is sent to the client for rendering. This strategy requires very high bandwidth and resources at client, thus may not be available in many applications. Alternatively, the volume can be rendered at the server and only the rendered result is sent to the client (*render remote*). Although relatively lower bandwidth is required, the inevitable network latency will degrade the overall performance.

An approach to sidestep the hurdle caused by the data size is to consider region-of-interests (ROI) or focus plus context visualization. Despite the large data size, interesting features are typically localized and appeared in a small ROI. Thus, it is acceptable to display objects in the ROI in full resolution, and omit details for objects outside the ROI. This leads to a reduction in information and thus, potentially, lowers down the computation and communication requirements. Several research works have studied on this multi-levels ROI rendering. Levoy et al. [9] gave a real-time volume rendering system that rendered volumes in two different levels of resolution. These two rendered images were then blended to obtain the final rendered image. Piccand et al. [11] described a method to perform X-ray projection in the wavelet domain, such that the ROI was projected in full resolution, while other voxels were projected in reduced resolution. Along a viewing ray that entered the ROI, voxels lying before or after the ROI were omitted



(a) (b)

Figure 1: (a) A full resolution data-set with 512x512x426 voxels. (b) Rendered image of a foveated volume. The fovea is at the red dot. A total of 5,435,144 coefficients are required to represent this foveated volume, which is a reduction to 5% with respect to the original volume.

in the projection. The main technique employed by Piccand et al. was wavelet splatting [8], which precomputed a 2D projected *footprint* for each sub-band.

From another perspective, an interactive visualization session can be more effective if the objects in the ROI are highlighted and information outside the ROI is filtered or reduced. Hence, even if the whole data-set is available, or there are sufficient computing resources, focus plus context visualization can still be useful. This leads to the issue of how to effectively visualize a volume with a point of focus. Zhou et al. [14] proposed using distance as a factor to adjust objects' opacity. Viola et al. [12] presented a technique that suppressed less important information in volume rendering by cutting away objects occluding the interesting objects.

In this paper, we employ the notion of foveation [5] to achieve different levels-of-detail for rendering. A foveated volume can be viewed as a non-uniform sampled volume, where the density of the samples is highest at a point (fovea). Figure 1(a) and Figure 1(b) show examples on the rendered images of a full resolution volume (512x512x426 voxels) and a foveated volume. An ideal foveated volume has a smooth transition from the highest to the lowest resolution. However, in practice, only a small number of levels is presented. In Figure 1(b), there are 4 different levels of resolution. The overall information in a foveated image/volume is significantly reduced, compared to the original data. Such reduction in information has been exploited in image transmission, video conferencing and computer vision systems [2, 3, 5, 6].

We give an algorithm that renders a foveated volume from its wavelet coefficients efficiently, such that the rendering time depends on the number of the wavelet coefficients. Specifically, if the foveated volume is represented in m wavelet coefficients, the algorithm runs in $O(n^2 + m)$ where n is the width in number of pixels of the rendered image. For simplicity, we consider one ray per voxel, hence n is also the width of the volume.

Note that previously known fast rendering algorithms do not fully exploit the reduction in information in the sense that, voxels that appear before or after the ROI are either omitted or rendered in high resolution. Our algorithm achieves speedup by tracking the "thickness" of the rays during rendering. The algorithm does not require preprocessing on the data (the forward wavelet transformation on the full volume is not considered as preprocessing). Hence, various viewing parameters, such as the transfer functions, can be interactively modified. To further exploit the characteristic of foveated rendering, we also propose two ways to visualize foveated volumes.

Outline. We first give an overview of foveation in Section 2.1 and some notations in Section 2.2. In Section 2.3, we describe the rendering equation used

in this paper. In Section 3 we present our algorithm. Section 4 describes two ways to visualize foveated volume. Section 5 discusses a method to reduce staircase artifacts of the rendered results. In Section 6, we present our experimental results. Section 7 gives some potential applications of our algorithm. Section 8 concludes the paper.

2 Background and Notations

2.1 Foveation

A foveated image can be viewed as a non-uniform sampled image, where the density of samples is the highest at the fovea, but falls off as the distance from the fovea increases. The human visual system has similar distribution of resolution. Chang et al. [5] describe a formulation of the "ideal" foveated image. Such notion can be easily extended to three dimensions. A foveated volume is obtained from a full resolution volume by a *foveation* process. This process depends on two parameters, the *fovea* x_0 which is a point in the 3D space, and *rate* r_0, a non-negative number. Given a volume V, foveation applies space-variant smoothing function on V. At locations nearer to the fovea, the width of the smoothing function is smaller. The rate r_0 determines how fast the width of the smoothing function grows.

One approach to approximate a foveated image while keeping data size small, is by retaining some coefficients in the wavelet domain. Let us describe the approximation using a 16x16 pixels image. Similar idea can be applied to volume. Figure 2(a) shows the retained coefficients for a foveated image. Coefficients in the shaded squares of Figure 2(a) are retained. If the image is represented using Haar wavelet, then the foveated image is as shown in Figure 2(b) where pixels in each box has the same value. Note that the widths of the shaded squares are the same except for those that touch the boundary. The location of each square with respect to the co-ordinate of the sub-band depends on the fovea location $\mathbf{x}_0$. The common width of the squares depends on the rate r_0. For convenience, we simply refer the width as rate. A better approximation can be achieved by using circles instead of squares, applying a weighting function on the coefficients, and having circles with slightly different size in different sub-bands [5].

2.2 Notations

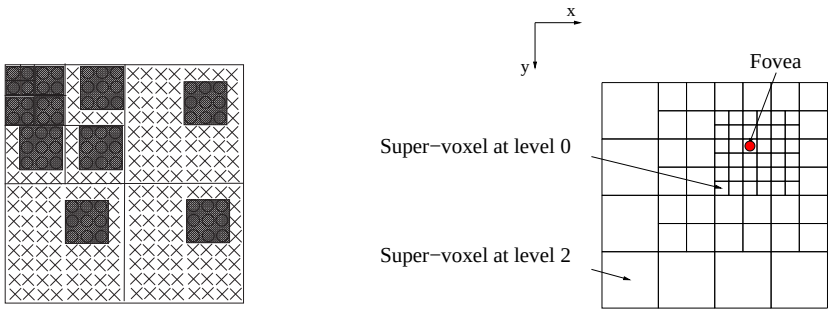
Co-ordinate system. Our volume data-set V is stored in a $n \times n \times n$ array. The indices of the array (starting from 0 to $n-1$) also serve as the locations of the voxels in the 3D space.

Same as in images, a wavelet coefficient of the three dimensional V is labeled by its sub-band and location. Unlike images, in three dimensions, there are seven high frequency sub-bands at each level. We use the convention that sub-bands with the coarsest resolution are defined to be at the 0-th level. Thus, performing forward wavelet transformation on the i-th level sub-band $\mathtt{LLL}_i$ gives a $(i-1)$-th level sub-band $\mathtt{LLL}_{i-1}$, and seven other high frequency sub-bands. We also assume each sub-band is stored in a 3D array and use its index to serve as the location of the wavelet coefficient. Hence, the spatial location (x, y, z) corresponds to $(x/4, y/4, z/4)$ in the sub-band $\mathtt{LLL}_{\log_2 n-2}$.

We call a coefficient in a low frequency sub-band $\mathtt{LLL}_\ell$ a *super-voxel* at level ℓ. Each super-voxel can be viewed as a cube in the spatial domain. A ℓ-th level coefficient at the location (x, y, z) (with respect to the co-ordinate in the sub-band) corresponds to a cube of width $n/2^\ell$ at $(n/2^\ell x, n/2^\ell y, n/2^\ell z)$ in the spatial domain.

Wavelet Foveation and super voxels. For convenience, we simply call the approximation of the "ideal" foveated volume the foveated volume. Recall that the approximation is done by selectively retaining some coefficients, and is parameterized by the location of fovea $\mathbf{x}_0$, and the rate r_0. We denote the foveated volume data as $V_\mathrm{f}(\mathbf{x}_0, r_0)$.

Let $C(\ell, \mathbf{x}_0, r_0)$ be the set of wavelet coefficients in the ℓ-th level high frequency sub-bands, and is contained in the cubes whose two opposite corners (with respect to the co-ordinate in the respective sub-band) are at



(a) (b)

Figure 2: Wavelet foveation. (a) $\mathbf{x}_0 = (10, 4), r_0 = 3$ in wavelet domain. (b) $\mathbf{x}_0 = (10, 4), r_0 = 3$ in spatial domain.

$$(n/2^\ell)\mathbf{x}_0 - \mathbf{r}', (n/2^\ell)\mathbf{x}_0 + \mathbf{r}' \tag{1}$$

where

$$\mathbf{r}' = \begin{pmatrix} \frac{r_0}{2} - 1 \\ \frac{r_0}{2} - 1 \\ \frac{r_0}{2} - 1 \end{pmatrix} and \ \ (r_0 >= 2).$$

The $C(\ell, \mathbf{x}_0, r_0)$ is in fact the ℓ-th level of wavelet coefficients retained for the foveated volume $V_\mathrm{f}(\mathbf{x}_0, r_0)$. Let $\mathcal{C}(\mathbf{x}_0, r_0) = C(0, \mathbf{x}_0, r_0) \cup C(1, \mathbf{x}_0, r_0), \ldots, C(\log_2 n - 1, \mathbf{x}_0, r_0) \cup \{w_0\}$ where w_0 is the only coefficient in $\mathtt{LLL}_0$. Hence, from $\mathcal{C}(\mathbf{x}_0, r_0)$, we can obtain $V_\mathrm{f}(\mathbf{x}_0, r_0)$ using inverse wavelet transformation.

Consider the coefficients in the sub-band $\mathtt{LLL}_\ell$, and are within the cube with the two corners given by (1). Each coefficient is a super-voxel, and let us denote these coefficients as $R(\ell, \mathbf{x}_0, r_0)$. The foveated volume $V_\mathrm{f}(\mathbf{x}_0, r_0)$ can be obtained by merging the super-voxels in $R(0, \mathbf{x}_0, r_0)$, $R(1, \mathbf{x}_0, r_0), \ldots, R(\log_2 n - 1, \mathbf{x}_0, r_0)$. Note that the total number of super-voxels is same as the number of wavelet coefficients in $\mathcal{C}(\mathbf{x}_0, r_0)$.

2.3 Volume Rendering Equations

An important volume data visualization technique is direct volume rendering. As light traverses through the volume it is emitted and absorbed. According to the optical model for direct volume rendering [10], the resulting intensity for the light along viewing rays to the viewer is given by

$$I(t_1, t_2) = \int_{t_1}^{t_2} V(t) e^{-\int_{t_1}^{t} \alpha(s) ds} dt \tag{2}$$

where t_1 and t_2 are the start and end points on the viewing ray, $V(t)$ is the intensity value at location t, and $\alpha(s)$ is the opacity at s.

In the discrete case, each sample in the volume is called a voxel. Equation 2 can be reduced to a finite sum over the accumulated opacity with the assumption that the intensity function and opacity function for a certain segment i are constants v_i and α_i. We yield

$$I = \sum_{k=1}^{n} v_k \alpha_k \prod_{i=0}^{k-1} (1 - \alpha_i) \tag{3}$$

3 Proposed Algorithm

Given the rate r_0, location of fovea $\mathbf{x}_0$, the wavelet coefficients $\mathcal{C}(\mathbf{x}_0, r_0)$ of the foveated volume, and the viewing parameters including the viewing direction θ and the transfer functions, we want to compute the rendered image of $V_\mathrm{f}(\mathbf{x}_0, r_0)$.

A straightforward algorithm solves the problem by first reconstructing the foveated volume $V_\mathrm{f}(\mathbf{x}_0, r_0)$

from $\mathcal{C}(\mathbf{x}_0, r_0)$ using inverse wavelet transformation, and next applying direct rendering on $V_{\mathrm{f}}(\mathbf{x}_0, r_0)$. This method is costly since representing $V_{\mathrm{f}}(\mathbf{x}_0, r_0)$ in the spatial domain already requires $\Omega(n^3)$ storage space. We give an algorithm that avoids reconstructing $V_{\mathrm{f}}(\mathbf{x}_0, r_0)$.

Our algorithm consists of two phases, reconstruction phase and rendering phase. In the first phase, given m wavelet coefficients of the foveated volume, the super-voxels $R(\ell, \mathbf{x_0}, \mathbf{r_0})$ is reconstructed. The reconstruction can be done in $O(m)$ time. In the second phase, the displayed image is rendered from the super-voxels. The rendering time is $O(m + n^2)$. These two phases can be combined to further reduce memory usage.

3.1 Main Idea

Rendering. Let us first describe the second phase which is more interesting. We will explain the rendering using a 2D example. We want to trace rays in a foveated image along the x-axis as shown in Figure 2(b), giving a 1D signal as output. In Figure 2(b), a lower resolution sample is depicted as a bigger square, which we call it a *super-pixel* (the analogous of super-voxel). Consider a set of rays tracing through a big super-pixel. If the intensities of the rays are the same before hitting the square, then they are also the same upon leaving the square. Thus, from computational aspect, all these rays can be emulated altogether in one step. Since they are the same, we group these rays into a *thick* ray, where the thickness is the width of the region it covers.

A key observation is that we can always *split* a thick ray, but not mix two rays. Consider the situation where a thick ray leaves a square and enters into two smaller squares. In this situation, the ray has to be split into two thinner rays. On the other hand, consider the situation where two adjacent thin rays, leave their respective squares and enter into a common bigger square. In this situation, the two rays may be different in intensity, when entering into the bigger square. Hence, no computation can be shared.

The darker arrows in Figure 3(a) show how the rays trace half-way through a foveated image. Due to the structure of foveation, we only need to split the rays. Problem arises in the second half of the foveated image if the rays continue to trace toward the right. Since rays can not be mixed, in the second half, they have to remain thin. This is not optimal since, intuitively, some computation could be shared in the second half. To overcome that, we trace the rays along x-axis in two directions, forward and backward as shown in Figure 3(b). The final rendered 1D signal is the composition of these two sets of rays. We do not set the line where the two sets of rays meet as a straight line, otherwise it may cut across a whole square.

For arbitrary viewing directions, we first apply shear-warp [7] on the super-voxels, and perform geometric correction on the rendered image.

Reconstructing super-voxels. Given the wavelet coefficients $\mathcal{C}(\mathbf{x}_0, r_0)$ of the foveated volume (the shaded squares in Figure 2(a)), we want to reconstruct the super-voxels. A full inverse wavelet transform will be costly. Fortunately, due to the special arrangement of those coefficients, the reconstruction can be restricted within a cube of width $(r_0 + s)$, where s is the wavelet support size. Thus the running time is in the same order as the number of selected coefficients. Such technique of achieving fast inverse transformation on foveated volumes is essentially same as the technique proposed by Yu et al. [13] for image rotation in the wavelet domain.

To further speedup, we can restrict reconstruction in the width of r_0, however, there will be a minor lost in accuracy. Note that it is possible to use wavelets with larger support, for example Daubechies 7/9 biorthogonal wavelets.

Total running time. The reconstruction phase takes $O(m)$ time where m is the number of wavelet coefficients. Also recall that the number of super-voxels is also in $O(m)$. For rendering, observe that the computation required is directly proportional to the number of rays, which is the number of super-voxels. Hence, the running time will be $O(m + n^2)$ where m is the total number of wavelet coefficients required, and n is the width of the rendered image.

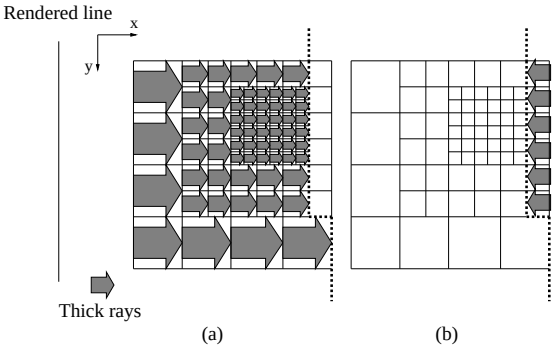


Figure 3: Thick rays rendering.

Combining reconstruction and rendering. If the volume is represented using Haar wavelet, it is possible to combine the reconstruction phase and rendering phase, so that the super-voxels are computed as and when required and not explicitly stored. In this way, additional memory space required can be reduced. In our implementation, we combine these two phases.

However, there are drawbacks in combining the two phases. The implementation would not be easy, especially when shear-warping is involved. Furthermore, it also removes some flexibilities. For instance, it is not clear how to extend it to wavelet with larger support when both phases are combined. In addition, in applications where the volume has to be rendered many times with the same coefficients but different viewing parameters, the reconstruction will be unnecessary repeatedly performed if the reconstruction and rendering phases are closely coupled.

4 Visualizing Foveated Volume

A foveated volume implicitly indicates that the interesting features are near the fovea. Hence, for effective visualization, it is desirable to give priority to the fovea, and to have a mean to direct the viewer's attention to the fovea. This can be achieved by multiplying the original opacity with a space-variant *weighting function*. Specifically, the opacity at location (x, y, z) is $T_\alpha(V(x, y, z))D_{\mathbf{x}_0}(x, y, z)$ where $V(x, y, z)$ is the voxel intensity, $T_\alpha(\cdot)$ is the opacity transfer function, $\mathbf{x}_0 = (x_0, y_0, z_0)$ is the fovea, and $D_{x_0}(\cdot)$ is the weighting function. Hence, the opacity of a voxel depends on both its location, and its intensity. We experiment with two weighting functions.

- The weighting function chops off all the voxels before the fovea along the viewing direction. If the viewing direction is along the x-axis, the function is:

$$D_{\mathbf{x}_0}(x, y, z) = \begin{cases} 1 & \text{if } x > x_0, \\ 0 & \text{otherwise} \end{cases}$$

  Figure 6(g) and Figure 6(h) show the effect of this weighting function.

- The weighting function varies across the 3D space. It is higher near the fovea, and its reciprocal increases linearly as the distance from the fovea increases. Specifically,

$$D_{\mathbf{x}_0}(x, y, z) = (1 + a\|\mathbf{x}_0 - (x, y, z)\|_2)^{-1}$$

  where $\|\cdot\|_2$ is the usual 2-norm and a is a constant that can be interactively adjusted by the viewer. When a is small, the variation across the space is lesser.

  Figure 5(b) shows the effect before applying the weighting function while Figure 6(a) shows the effect after it is applied.

5 Reducing Staircase Artifacts

The staircase artifacts (that is, the "blockish" effect) in the rendered image are due to the notion of thick ray in sharing computation. A way to reduce the artifacts is by post-processing. We can view the output of the rendering as a collection of non-uniformly spaced samples of thick rays, and the rendered image is the interpolation of these samples. The staircase artifacts appear when the sampling function is a step function. Alternatively we can use a smoother sampling function to reduce the artifacts. This can be done by performing a space-variant smoothing process on the original rendered image, where the width of the smoothing function is larger for thicker rays. Interestingly, the space-variant smoothing process is

essentially a foveation operation, which can be accurately and efficiently approximated using wavelet foveation (Section 2.1).

6 Experimental results

Rendering results. To evaluate the visual effect of our proposed algorithm, we applied our method on the CT scan of the visible man's torso with 512x512x426 voxels. Figure 4 shows the rendering on the full resolution volume as the ground truth. The rendering results of foveated volumes are given in Figure 5, all at a viewing angle of 30 degree.

Figure 5(a), Figure 5(b) and Figure 5(c) show the rendering with fovea parameters $\mathbf{x}_0 = (155, 353, 300)$ and rate r_0=100, 50 and 25. The fovea is marked as a red dot. Note there are staircase artifacts around the peripheral. The artifacts are reduced after a space-variant smoothing is applied as shown in Figure 5(d), Figure 5(e) and Figure 5(f).

Figure 5(g), Figure 5(h) and Figure 5(i) show rendering results on the same data set as Figure 5(a), Figure 5(b) and Figure 5(c), except that the fovea is moved to a new location.

Comparing Figure 4 and Figure 5, it is noted that no information is lost at the fovea while a large amount of coefficients are omitted for rendering foveated volumes.

Figure 6 shows the rendering results of applying weighting function mentioned in Section 4. After applying the second weighting function on Figure 5(b), we have Figure 6(a). Compared to the one without weighting function, the peripheral region appears darker. This is because the weighting function further suppresses information far from the fovea. Figure 6(b) employs a larger weighting parameter a than Figure 6(a). By this adjustment, the information about peripheral region is much more suppressed. Figure 6(c) gives result when the rate $r_0 = 25$. To reduce staircase artifacts, Figure 6(d), Figure 6(e) and Figure 6(f) apply smoothing on Figure 6(a), Figure 6(b) and Figure 6(c).

Figure 6(g) and Figure 6(h) show the chopping off effect- the first weighting function mentioned in Section 4. The full resolution volume for Figure 6(g) and Figure 6(h) is 256x256x225 voxels and the rate r_0 is 40. Note that each of these images is not simply an image of a plane slicing through the volume. This can be observed in Figure 6(h), where the surface of the ear is vaguely visible.

Table 1: Comparison of frame rates on different data-sets. The viewing direction is along x-axis for direct volume rendering. Note that VolPack requires large preprocessing time. Due to the memory limit of our machine, we only compare these three methods on these small size data-sets. In Figure 7(c), we give the performance analysis of our algorithm on larger data-sets.

Data-set	Direct volume rendering (Frame rate)	VolPack (Frame rate/MV MO/CV)	Our alg. (Frame rate)
brain	32.3	106.4/0.91 0.03/0.16	43.5
engine	6.4	45.9/4.66 0.19/0.69	42.6
head	3.2	16.0/9.09 0.33/1.67	25.6

Computational Performance. To evaluate the performance of our algorithm, we tested our method on three data-sets: a MRI scan of a head with 128x128x84 voxels, a CT scan of an engine with 256x256x110 voxels, a CT scan of a human head with 256x256x225 voxels. These are described as "brain", "engine" and "head" in Table 1. All experiments were conducted on a 3GHz Pentium IV PC with 1GB DDR RAM.

We compared our proposed algorithm with the VolPack volume rendering library [1], and a straightforward direct volume rendering. Table 1 gives the frame rate (in Hz) and rendering time (in seconds) on the test data-sets by different rendering methods. Note that the large 512x512x426 voxels volume is not tested on VolPack since VolPack is unable to process the large volume under our machine configuration.

In direct volume rendering, the rendering equation is applied to the full resolution volume using the straightforward for-loops, with the viewing direction along the x-axis.

There are three rendering algorithms provided by VolPack. The fastest algorithm relies on a special data structure containing run-length encoded, classified volume data. Preprocessing is required to obtain this data structure. Hence, it is suitable for rendering the same volume without changing classification. MV, MO and CV represent the three preprocessing steps, which are:

- Make volume (MV): Create an unclassified volume from the raw volume data. This unclassified volume includes precomputed information for shading and classification;
- Make octree (MO): Create a min-max octree from the unclassified volume;
- Classify volume (CV): Create a classified volume including an opacity with each voxel along with shading information.

VolPack provides accurate classification. Even if we just consider MO which deals with the octree and the structure of resolution, the preprocessing time is still non-negligible.

If the volume is already represented by its wavelet coefficients, no preprocessing is required for our algorithm. Hence, if the viewer wishes to interactively change the transfer functions for the intensity and opacity, our algorithm is still able to give real time feedback for large data-set. The foveation parameters we use in Table 1 are r_0=25, $\theta = 45$ degree.

Figure 7 gives rendering time for different viewing parameters and data width.

Figure 7(a) shows that the rendering time increases as the fovea rate r_0 increases. It is because as r_0 increases, more wavelet coefficients are selected for rendering. For the same r_0, the time for rendering with viewing angle at 45 degree is larger than that at 0 degree since there are more data to be processed in the shear-warp operation. The reason is also true for Figure 7(b) which shows that the rendering time increases as the viewing angle.

Figure 7(c) shows that the rendering time increases as the data width increases. When n is large, the time is proportional to n^2 as m is small. When the width increases from $n = 1024$ to 4096, although the data size increases by a factor of $4^3 = 64$, the rendering time only increases by approximately a factor of 6.86 when the viewing angle θ is 45 degree.

7 Applications

A potential application of our algorithm is in remote volume visualization. A viewer at the client-side indicates the fovea, and the selected coefficients are sent across (alternatively, we can let another viewer at the server-side indicates the fovea). In the client-side, the viewer applies our algorithm to render the obtained foveated volume. The server continues to send coefficients across, achieving the effect that the fovea rate is increasing. For the viewer, what he/she sees is the rendering result that is getting more and more accurate. Note that if direct rendering method is used here, then the inverse wavelet transformation has to be applied for every new coefficient arriving at the client-side. Our algorithm works efficiently in the wavelet domain and hence overcomes this problem.

Another application is in the visualization of time-varying volume data. If the time-varying volume data is already represented in a foveated form, it is possible to apply our idea to achieve fast rendering. For example, consider a time-varying volume data which are the output of a set of sensors. The distribution of the sensors in the 3D space resembles the structure of foveated volume, with higher density around a fovea. The coverage of the sensors could be wide and thus impossible to perform a full-resolution real-time volume rendering. Our algorithm is a possible solution.

8 Conclusion

In this paper, we presented an algorithm that renders a foveated volume efficiently in the wavelet domain. The required running time for rendering the foveated volume is $O(n^2 + m)$ where n is the width of the ren-

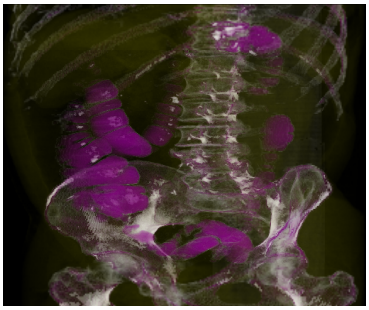


Figure 4: Rendering of a full resolution volume having 512x512x426 voxels with viewing angle $\theta = 30$ degree.

dered image, and m is the number of retained wavelet coefficients. We implemented the algorithm and analyzed its performance. The experimental study also confirmed the efficiency of the algorithm, even for very large n. Excluding the forward wavelet transformation, no expensive preprocessing is required on the original volume. Compared to the rendering of the full resolution volume, our method produces the image with the same quality at the fovea but lower resolution further way. The method provides a good tradeoff between rendering resolution and frame rate. It is suitable to be applied in scenarios where the rendering platform has low computing resources and/or real time feedback is required.

References

[1] The VolPack volume rendering library. http://graphics.stanford.edu/software/volpack/ (1995)

[2] Basu, A., Sullivan, A., Wiebe, K.: Variable-resolution teleconferencing. IEEE System, Man, and Cybernetics (1993)

[3] Basu, A., Wiebe, K.: Videoconferencing using spatially varying sensing with multiple and moving fovea. IEEE Trans. on Systems, Man and Cybernetics (1998)

[4] Bethel, W., Tierney, B., Lee, J., Gunter, D., Lau, S.: Using high-speed WANs and network data caches to enable remote and distributed visualization. In: Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), pp. 59–59 (2000)

[5] Chang, E.C., Mallat, S., Yap, C.: Wavelet foveation. Journal of Applied and Computational Harmonic Analysis (2000)

[6] Colombo, C., Rucci, M., Dario, P.: Integrating selective attention and space-variant sensing in machine vision. Jorge L.C. Sanz, editor, Image Technology: Advances in Image Processing, Multimedia and Machine Vision pp. 109–128 (1996)

[7] Lacroute, P., Levoy, M.: Fast volume rendering using a shear-warp factorization of the viewing transformation. Computer Graphics (1994)

[8] Laur, D., Hanrahan, P.: Hierarchical splatting: a progressive refinement algorithm for volume rendering. SIGGRAPH Comput. Graph. **25**(4), 285–288 (1991)

[9] Levoy, M., Whitaker, R.: Gaze-directed volume rendering. In: Computer Graphics (1990)

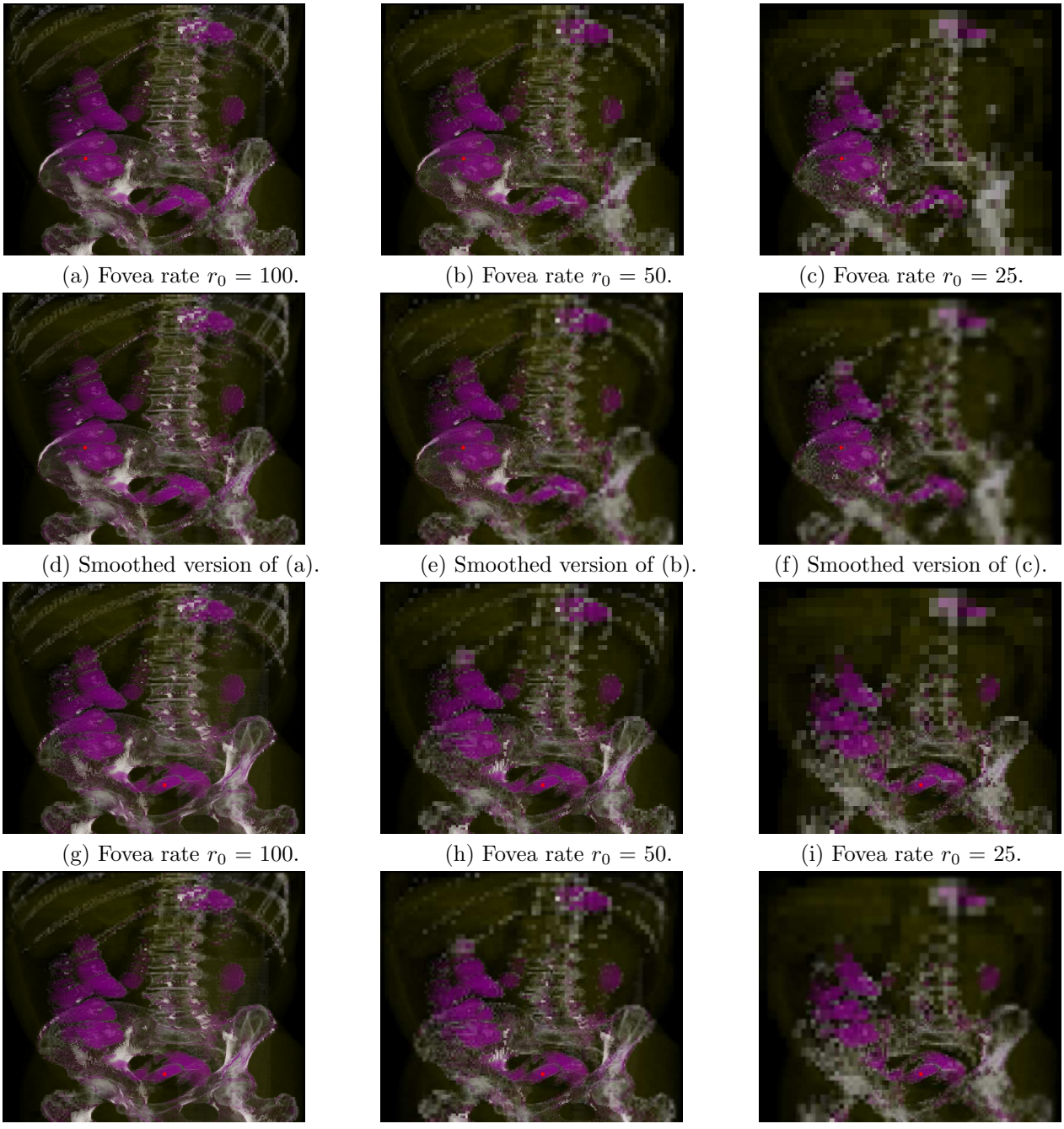
[10] Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics **1**(2), 99–108 (1995)

[11] Piccand, S., Noumeir, R., Paquette, E.: Efficient visualization of volume data sets with region of interest and wavelets. In: SPIE Medical Imaging (2005)

[12] Viola, I., Kanitsar, A., Gröller, M.E.: Importance-driven volume rendering. In: Proceedings of IEEE Visualization'04, pp. 139–145 (2004)

[13] Yu, H., Nguyen, V.T., Chang, E.C.: Rotation of foveated image in the Wavelet domain. IEEE International Conference on Image Processing (2004)

[14] Zhou, J., Döring, A., Tönnies, K.D.: Distance based enhancement for focal region based volume rendering. In: Proceedings of Bildverarbeitung für die Medizin 2004, pp. 199–203. Berlin (2004)



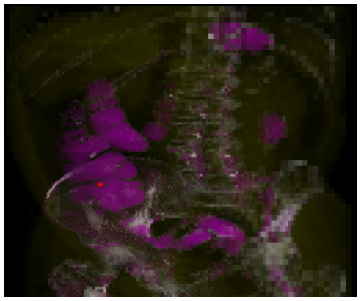
(a) Fovea rate $r_0 = 100$. (b) Fovea rate $r_0 = 50$. (c) Fovea rate $r_0 = 25$.

(d) Smoothed version of (a). (e) Smoothed version of (b). (f) Smoothed version of (c).

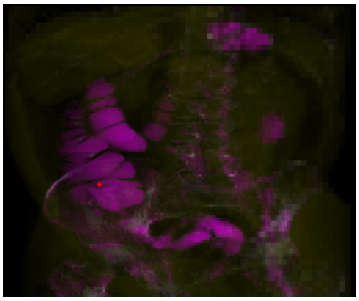
(g) Fovea rate $r_0 = 100$. (h) Fovea rate $r_0 = 50$. (i) Fovea rate $r_0 = 25$.

(j) Smoothed version of (g). (k) Smoothed version of (h). (l) Smoothed version of (i).

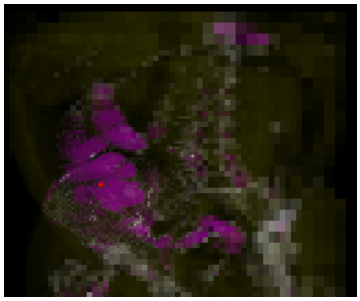
Figure 5: This set of images demonstrates the effect of fovea rate and location on the foveated volume. The rate is indicated below each image. Each image in the second and fourth rows is the smoothed version of the image above it. The number of coefficients retained for the foveated volume with rate 100, 50 and 25 is approximately $23.7\text{x}10^6$, $5.6\text{x}10^6$ and $0.9\text{x}10^6$ respectively. This amounts to a reduction to 21.3%, 5% and 1% of the original volume. The first two rows and the last two rows have different fovea location. The fovea is marked as a red dot in each image.



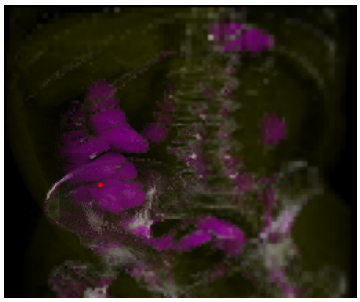
(a) Fovea rate $r_0 = 50$.



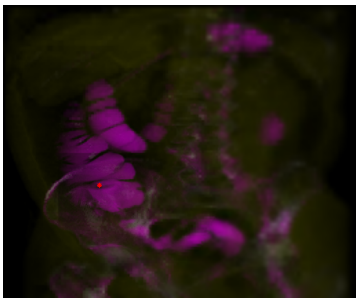
(b) Fovea rate $r_0 = 50$.



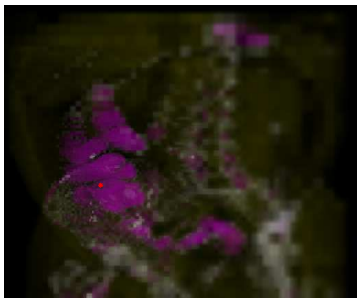
(c) Fovea rate $r_0 = 25$.



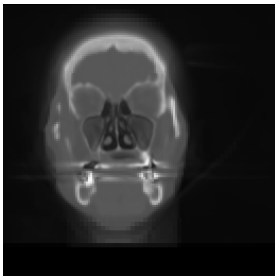
(d) Smoothed version of (a).



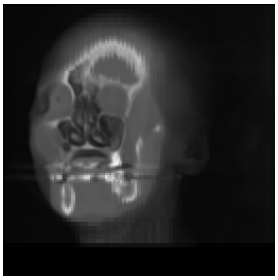
(e) Smoothed version of (b).



(f) Smoothed version of (c).

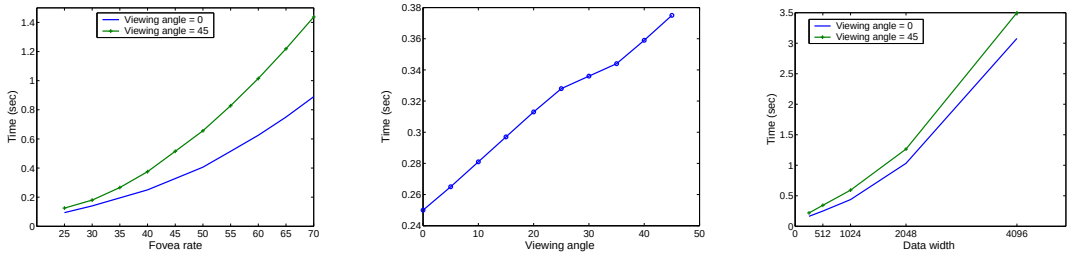


(g)



(h)

Figure 6: This set of images illustrates the effect of weighting function in visualizing foveated volume. (a) Rendering with varying opacity with fovea rate $r_0 = 50$. (b) Rendering with varying opacity with a larger constant a, compared to the rendering in (a). (c) Rendering with varying opacity with fovea rate $r_0 = 25$. (d) Smoothed version of (a). (e) Smoothed version of (b). (f) Smoothed version of (c). (g) The effect by chopping off the region before the fovea with viewing angle at 0 degree. (h) Same effect as (g) with viewing angle at 30 degree.



(a) (b) (c)

Figure 7: Rendering time. (a) Rendering time versus the rate r_0. (b) Rendering time versus viewing angle. The original volume for (a) and (b) has 512x512x426 voxels. Observe that more computation is required for larger angles. The worst case performance occurred at 45 degree. Here the rate is 40. (c) Rendering time versus data width. More time is required for a viewing angle at 45 degree.