Time-Critical Multiresolution Volume Rendering using 3D Texture Mapping Hardware

Xinyue Li and Han-Wei Shen
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
E-mail: xli@cis.ohio-state.edu and hwshen@cis.ohio-state.edu

Abstract

This paper presents a LOD selection algorithm for rendering multiresolution volumes using 3D texture mapping hardware. The main focus of our algorithm is to automatically select appropriate LODs from the volume hierarchy so that the rendering is completed within the user specified frame time. We use an intra-frame predictive-reactive scheme which does not require a precise performance model nor suffers from frame rate oscillations. In addition, our algorithm has a direct control on how the availble rendering time is distributed among the volume subdomains to ensure that important regions are appropriately emphasized. Our experiments showed that we are able to control the rendering time within a 10% difference from the user specified time budget.

1 Introduction

Although the speed of volume rendering has significantly increased in the last several years, the size of an average volumetric data set also continues to grow. To address the challenges imposed by the large data sets, intensive research efforts have been undertaken. Among the available methods, hierarchical rendering algorithms have shown a great potential. In essence, hierarchical methods first create a multiresolution representatoin for the underlying volume. Data of different resolutions in different regions are then chosen at run time to perform rendering. The goal of the algorithms is to adaptively simplify the rendering in regions where data are relatively unimportant or uninteresting, so that both the memory consumption and computational cost are reduced without significantly affecting the rendering quality.

While the hierarchical algorithms[1][2][3] can accelerate the speed of rendering, the selection of appropriate volume resolutions, or level of detail (LOD), for effective quality speed tradeoff is often done on an ad-hoc basis. Frequently used approaches such as selecting the volume resolutions based on the volume block's distance to the view point and the angle to the viewing vector [1][2], or based on user specified error tolerances [4] cannot effectively control the rendering time or quality. Although LOD selection algorithms [5][6] for polygon rendering systems have been intensively studied in the past, LOD selection algorithms for volume rendering, however, are still scarce. This is primarily because not until recently cost-effective 3D texture mapping hardware used for interactive volume rendering became widely available. However, in order to successfully apply volume rendering to time-critical applications such as virtual surgery[7] or real time volume navigation[8], the ability to perform time-critical volume rendering with flexible level of detail control is now becoming increasingly important.

This paper presents a LOD selection algorithm for rendering hierarchical volumes using 3D texture hardware. The main focus of our algorithm is to automatically select appropriate LODs from the volume hierarchy so that the rendering is completed within the user specified frame time. Unlike the previous time-critical polygon rendering systems which perform time-critical control using static heuristics[9][10][11], inter-frame feedback[12][13], or global optimization[5][6] , we use an intra-frame predictive-reactive scheme which does not require a precise performance model nor suffers from frame rate oscillations. In addition, our algorithm has a direct control on how the available rendering time is distributed among the volume subdomains to ensure that important regions are appropriately emphasized. Our experiments showed that we are able to control the rendering time within a 10% error of the user specified time budget.

The rest of the paper is organized as follows. In section 2, we discuss related work on time-critical rendering algorithms. In section 3, we give an overview of our multiresolution rendering method. In section 4, we present our time-critical volume rendering algorithm in detail. Experimental results are discussed in section 5. Conclusion and future work are discussed in section 6.

2 Background

Previously researchers have proposed various run-time LOD selection methods for scenes consisting of polygonal models. Based on the selection strategies, existing methods can be classified into three categories: static heuristics, inter-frame feedkback, and global optimization. Algorithms based on static heuristics[9][10][11] determine LODs using fixed criteria such as object distance, view angles, or screen coverage. Feedback control algorithms[12][13] adjust the LODs according to the difference between the desired and the actual rendering time from the previous frame. Global optimization algorithms [5][6], on the other hand, rely on rendering performance models and object benefit heuristics. The objective of the optimization methods is to maximize the image benefit while limiting the rendering cost to the user-specified frame time.

Generally speaking, static heuristics cannot precisely control the rendering time because the relationship between the exact rendering time and the heuristics in use is hard to find. Inter-frame feedback schemes can suffer from frame-to-frame rendering time oscillation and thus can be ineffective. Adopting the global optimization methods for LOD selections can be difficult in practice because a precise performance model for the rendering pipeline is difficult to get. Many dynamic factors such as memory page fault and system call delays will affect the rendering performance but these factors are difficult to model. In addition, for scenes with a large number of textures and when the total size of the textures is larger than the texture memory capacity, it is non-trivial to predict in advance whether a particular texture will be resident in the texture memory or not when it

is needed.

To control the run-time performance of volume rendering, previously we proposed an automatic error tolerance specification system for hardware volume rendering using Fuzzy Logic Control [12]. The system was able to dynamically track the performance of rendering and adjust the error tolerances on the fly to satisfy the user's performance goal. While effective, the algorithm has several drawbacks. First, the algorithm is based on feedback control, which can suffer from the frame time oscillation problem. The second problem of our previous algorithm is that there is no explicit control on how the time should be spent to render different portions of the volume. For instance, given a fixed amount of computation time, the user may want to render the volume in a medium resolution uniformly. Or, the user can use the same amount of time to render certain regions in a higher quality and tolerate a lower quality in other regions. Our previous algorithm, as well as most of the time-critical algorithms, do not allow such a flexible control. In this paper, we propose a time-critical algorithm for hierarchical volume rendering that can both guarantee the rendering frame rate and allow a flexible control in the rendering quality.

3 Multiresolution Volume Rendering

Before talking about our time-critical volume rendering algorithm, we first give a brief overview on how we build our multiresolution volume hierarchy and how we perform the multiresolution rendering.

3.1 Multiresolution Volume Construction

We build our hierarchical volume structure using an algorithm similar to the one proposed by Weiler *et al.*[2]. Initially, the entire volume space is subdivided into smaller blocks, hereafter called subvolumes. For each subvolume, we create different levels of detail by repeatedly filtering the voxel data contained in the subvolume. Starting from the raw data, we average every $2 \times 2 \times 2$ voxels to create a lower resolution subvolume. We continue this filtering process until a predefined minimum resolution for the subvolume is obtained. To avoid seams between adjacent subvolumes of different resolutions, we adopt the method proposed by Weiler *et al.*[2], which copies data points on the boundaries from low resolution to high resolution volumes to ensure a smooth transition when interpolation is performed. More details about the data filtering and seam prevention can be found in [2].

Our algorithm differs from the method proposed by Weiler *et al.*[2] in that the subvolumes in our case have different sizes. Instead of subdividing the volume uniformly into subvolumes of equal size, we take into account the volume's spatial coherence when performing the subdivision. In regions where data values are more coherent, we merge the voxels together to form a larger subvolume. On the other hand, if the data values in a region have higher variations, we split the region into smaller subvolumes. There are two primary reasons for us to use subvolumes of different sizes. First, breaking a volume into subvolumes creates overhead since more slicing planes are needed when rendering the volume. Volume data that have high spatial coherence can usually be rendered using lower resolution data so breaking a larger subvolume into smaller ones is unnecessary. On the other hand, if the volume contains values that are less coherent, breaking the volume into smaller subvolumes can allow us to use higher resolution data in those interesting but local regions while using low resolution data elsewhere.

We implement this adaptive subdivision of the volume using a method similar to the algorithm proposed by Laur and Haranhan[4]. A complete octree is first created in which each tree node records the standard deviation of the voxel data within the corresponding region. The subvolumes are then formed by using a user-specified threshold to traverse the octree. Starting from the root of the octree, a splitting of the volume into eight subvolumes is performed when (1) the size of the subvolume is larger than the texture memory (2) the standard deviation of the subvolume is higher than the user-supplied threshold. Once the subvolumes are obtained, we perform the multiresolution filtering as mentioned above.

3.2 Multiresolution Rendering

The final rendering image is produced by rendering individual subvolumes in front to back order and blend the results together. For each subvolume, we first select an appropriate LOD, then use a standard 3D texture mapping algorithm[14] to perform the rendering. Specifically, the texture hardware rendering algorithm consists of computing a sequence of slicing polygons perpendicular to the view vector within the subvolume, and then use OpenGL 3D texture mapping and alpha blending functions[15] to shade and blend the polygons together. Data of the selected LOD for each subvolume need to be loaded into the texture memory before the subvolume is rendered. To minimize the texture creation and loading overhead, we create an OpenGL texture object for each of the different LODs for every subvolume before the rendering starts. During the rendering stage, a texture object bind corresponding to the selected LOD data for each subvolume is performed. The use of OpenGL texture objects allows the reuse of existing textures, which is much faster than reloading the texture data using glTexImage*D(). In addition, if the underlying implementation of OpenGL supports texture working set, transmissions of some texture objects from main memory to the texture memory can be completely avoided.

4 Time-Critical LOD Volume Rendering

Given the different LODs for each subvolume, the goal of our time-critical volume rendering algorithm is to select an appropriate LOD for each subvolume at run time so that the overall rendering time will be close to the user-specified frame time. In addition, to make sure that regions of interest are appropriately emphasized, our algorithm allocates the rendering time to different subvolumes based on the user-defined importance values, which can be simple heuristics such as the distance to the view point, the volume opacity, or other application-specific criteria. Based on these requirements, we propose an intra-frame predictive-reactive algorithm to perform time-critical volume rendering. In the following, we describe our algorithm in detail.

4.1 Intra-Frame Prediction Reaction

Our algorithm takes into account the run-time behavior of the rendering program when determining the LODs for the volume. As mentioned previously, the entire volume is subdivided into a list of subvolumes, which are rendered in a front-to-back sequential order. Instead of making a global decision about the LOD for every subvolume at once, the LOD decision for each subvolume is deferred until the subvolume is ready to be rendered. In our algorithm, the LOD for each subvolume is determined based on the following run time characteristics. First, performance statistics for rendering the previous subvolumes are gathered, which are then used to predict the rendering times for the different LODs of the next subvolume. Second, we keep track of the amount of time that is left, and the relative importance values among the remaining subvolumes. The importance of a

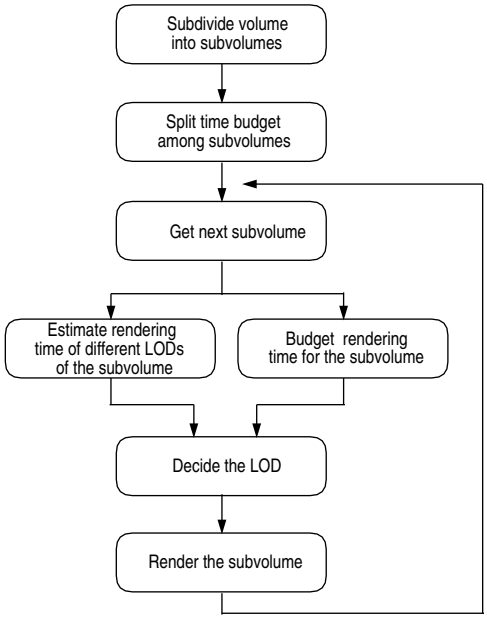


Figure 1: Algorithm Overview

subvolume is determined by the user-specified criteria. Finally, we allocate a share of the remaining rendering time to the next subvolume based on its relative importance, and then choose one of the LODs that has the closest predicted rendering time to the allocated time budget. We repeat this process for every subvolume until the rendering of the entire volume is complete. Figure 1 gives an overview of our algorithm.

Our algorithm differs from the existing LOD selection algorithms in several aspects. First, the LOD determination is performed at an intra-frame level in an adaptive manner, which makes it more effective in avoiding frame to frame rendering time oscillation. Second, the dynamic control of rendering speed does not totally depend on having an accurate performance model, which makes our algorithm more robust in practice. Third, we are able to control the rendering time allocated to each subvolume in a more direct manner, which makes it easier to emphasize regions of interest. In the following, we provide an in-depth discussion of several important components in our algorithm.

4.2 Rendering Time Prediction

As described above, we need to estimate the rendering cost for each of the LODs associated with a subvolume at run time. Instead of completely relying on a static performance model to perform the prediction, we take into account the system's run-time behavior by continuously collecting the performance statistics from the previously rendered subvolumes to assist the process. We use the following performance model to describe the rendering time T for a subvolume at a particular LOD using texture hardware:

$$T = t_{texture_loading} + max \begin{cases} t_{geometry} \times n_{slices} \\ t_{rasterize} \times n_{slices} \end{cases}$$

where $t_{texture_loading}$ is the time to load the volume texture of the LOD, $t_{geometry}$ is the average geometry processing cost to render a slice within the subvolume (transformation, clipping, per vertex lighting, etc.), $t_{rasterize}$ is the average rasterization cost to render each slice, and n_{slices} is the number of slices used for the LOD in question. Modern graphics hardware implements geometry processing and rasterization in a pipelining manner. Therefore, the overall rendering time for a graphics object is dependent on the bottleneck of the pipeline, which is the maximum of the geometry processing and the rasterization time. Although the slicing polygons in a subvolume have different sizes and shapes, we can calculate the average cost per slice by dividing the processing time by the number of slices. We note that in this performance model we assume that the same average geometry processing time and average rasterization time per slicing polygon for different LODs in a subvolume. This assumption is based on our observation that the slicing polygons in the subvolume of a particular LOD are evenly distributed in space. When we change the LODs, although the number of slices is reduced, the new set of slicing polygons still spread evenly across the entire subvolume. Therefore, similar geometric characteristics such as the average size and the average projection area will be kept. Hence, the per slice average geometry processing and rasterization cost remain approximate the same.

To use the above performance model in practice, we need to estimate the values of the performance variables used in the formula. Specifically, given a LOD of a fixed size, we need to know the texture loading time, the geometry processing time, and the rasterization time. Unlike the previous methods[5][6] which use static profiling, we compute the values based on dynamic profiling. That is, we collect the performance statistics based on the previously rendered subvolumes and estimate the rendering time for the new subvolumes. In essence, our run-time profiling and prediction consist of two parts: texture loading time prediction and slice rendering time prediction. In the following we discuss these two parts in detail.

4.2.1 Predict the texture loading time:

We assume that the texture loading time is in proportion to the number of voxels in the LOD. That is, it can be written as:

$$t_{texture_loading} = (t_{avg_voxel} * n_{voxel}) * (1 - R)$$

Where t_{avg_voxel} is the average time to load each voxel, n_{voxel} is the number of voxels in the LOD, and R is a boolean value depicting whether the texture is already resident (R = 1) in the texture memory or not (R = 0). In our implementation, to minimize the run time texture loading overhead we create a texture object for each of the LODs associated with a subvolume at the rendering program's initialization stage. To use a particular texture object during rendering, we use the openGL glBindTexture() command to activate the texture. Since multiple texture objects can be resident in the texture memory if the texture working set feature is supported by the underlying graphics hardware, we need to know first whether the texture object is resident in the texture memory when it is activated before determining the texture loading time, i.e., to find the value of R. This can be achieved by calling glAreTexturesResident() to query the residency of all the subvolume's LOD textures simultaneously.

To find the per voxel loading time t_{avg_voxel}, we perform profiling in the initialization stage of the program. After the texture objects are created, we call glBindTexture() for every texture object and force the graphics hardware to load the texture data to texture memory by drawing a dummy polygon using each of the textures. After all the textures are processed, we divide the total time taken by the number of voxels to get per voxel loading time.

4.2.2 Predict the slice rendering time

To predict the rendering time for the subvolume slices, we need to know the per slice average geometry processing time $t_{geometry}$, and the average rasterization time $t_{rasterize}$. In theory, we also need to know exactly whether the rendering is geometry limited or rasterization (fill) limited for a given viewing parameter. However, since in our performance model both the total geometry processing time and the rasterization time for a LOD of a subvolume is linearly proportional to the number of slices in the subvolume, and the sizes of subvolumes in the volume do not vary dramatically, given the same viewing parameters, we can assume that all the subvolumes have similar rendering characteristics, i.e, they are either all geometry limited, or all fill limited under the same viewing parameters. Based on this assumption, we only need to know the average processing (geometry or rasterization) time for each slice for the given viewing parameters at run time, and do not need to know whether the current rendering is geometry or fill limited. Hence, we can modify the performance model to:

$$T = t_{texture_loading} + t_{processing} * n_{slices}$$

where $t_{processing}$ is the average processing time for each slice. We obtain this number through the run-time, that is, we accumulate the slice rendering time for all the subvolumes that have been rendered (excluding the texture loading time), and then dividing it by the total number of rendered slices so far. The resulting average processing time is then the $t_{processing}$, and can be used to predict rendering time for the next subvolume.

4.3 Time Budget Allocation

A unique feature of our time-critical algorithm is that it is able to take into account the importance of the subvolume, and to assign different time budget for different subvolumes based on their importance, that is, we will spend less time to render the unimportant subvolumes, and the time saved is devoted to more important subvolumes so that it is possible to render them in a higher quality. Our time budget allocation algorithm is done by using an importance function supplied by the user. The importance function defines an importance value for each subvolume and can be chosen based on the particular need of the underlying application. Assuming an importance function is available, our time budget allocation algorithm works as follows. At the end of rendering each subvolume, we recalculate the available time T_b that is left, and then redistribute it to the remaining k subvolumes $v_j, j = 1..k$. Assuming that each subvolume v_i has an importance value I_i, our algorithm calculates the time budget t_i for the subvolume as:

$$t_i = T_b \times \frac{I_i}{\sum_{j=1..k} I_j}$$

With the above equation, we are able to give a larger amount of time budget to a subvolume with a higher importance value. The importance parameters used in our algorithm include:

- **Maximum opacity:** The maximum opacity of a subvolume is determined by the highest opacity of all the voxels in the subvolume. The rationale behind is that a more opaque region in the volume should be rendered in a higher accuracy. Therefore, a larger amount of time budget should be given.

- **Distance to the view point:** Here the distance is calculated from the center of the subvolume to the view point and the reciprocal of the distance is used as the importance function. For those subvolumes that are closer to the view point, as they have high importance value, they will be assigned more time budget and rendered in a high quality. For other subvolumes, as they are farther away and may be occluded, a lower time budget will be given.

- **Projection area:** Projection area is calculated based on the bounding box of a subvolume. For those subvolumes with high projection area, as we need more time to perform scan conversion, a higher time budget should be assigned to them.

- **Gaze distance:** This parameter is specially useful for gaze-directed rendering. Gaze distance is the distance between the center of the gaze area and the center of the projected area of a subvolume. For those regions that are closer to the gaze area, we assign more time budget to generate higher image quality. For those regions that are farther away from the gaze area, lower time is assigned and the image becomes blur.

We note that by appropriately assigning the time budget, we can easily modify the behavior of the volume renderer.

4.4 LOD Selection

The decision on the appropriate LOD for a subvolume is based on its budgeted rendering time and the predicted rendering time for all the possible LODs. The selection algorithm works as follows:

- Query the texture residency for all the LOD textures.

- Calculate the average processing time per slice based on the history as described previously.

- Calculate the predicted rendering time for each LOD of the subvolume using the performance model described above.

- Choose the LOD which has an predicted rendering time closest to the budgeted rendering time.

Since each subvolume typically only has 4-5 different LODs, the process of estimating the rendering time for all the LODs, and thus the LOD selection, can be done very fast. Therefore, the control overhead is very minor.

4.5 Temporal Coherence Consideration

The above LOD selection algorithm will be executed for every frame, which could cause the subvolume's LOD changes frequently. However, frequent change of volume LODs can cause the image to flick, which can be an undesirable effect. To solve this problem, we take special care to maintain the temporal coherence in consecutive rendering frames. We realize temporal coherence by recording the importance value of each subvolume from the last rendering. When a new frame is being rendered, if the user changes viewing direction or gaze area, the importance value of each subvolume will be recalculated. For those subvolumes whose importance value does not differ too much (we use a threshold of percentage to decide this), we do not change its LOD when rendering this frame, and thus maintain a good temporal coherence.

5 Results and Discussion

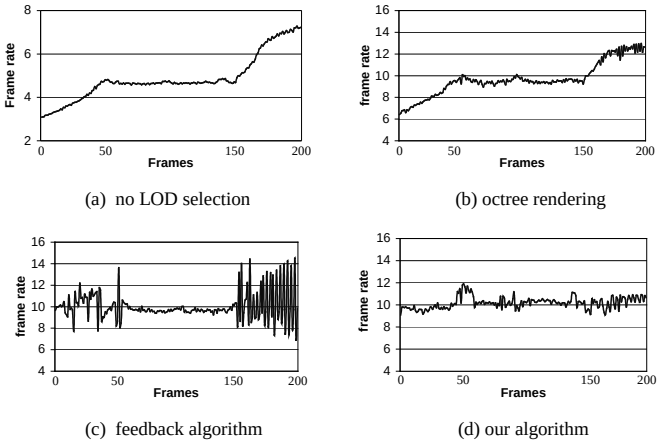
We have implemented our time-critical LOD volume rendering algorithm on an SGI Octane workstation with 4M bytes of texture memory. Two data sets were used for the experiments. One is a delta wing data set with a size of $222 \times 253 \times 103$; the other is a MRI brain data set with a size of $256 \times 256 \times 145$. Both of the data sets are in the floating point format. OpenGL's texture color lookup feature was used in our rendering program.

5.1 Guarantee the Desired Rendering Speed

In this section, we compare our time-critical algorithm with the following three methods:

- No LOD selection: each subvolume is rendered using the highest resolution.
- Octree based volume rendering algorithm: this method is similar to the algorithm used by Laur and Hanrahan[4]. A fixed error tolerance was specified to traverse the octree and each subvolume is rendered in a resolution corresponding to the level that the traversal stops. In this algorithm, different subvolumes are rendered using different resolutions based on their standard deviations.
- Feedback control algorithm: This algorithm is based on our previous work[12], in which we adjust the error tolerance based on the difference between the rendering time of the previous frame and the target frame time.

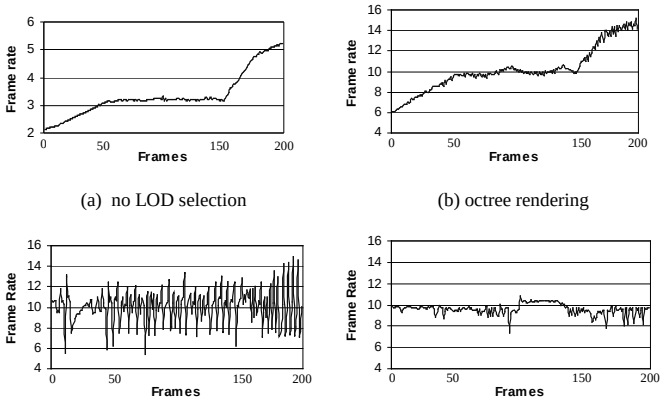
Figure 2 and figure 3 show the comparison of the rendering speed of the four algorithms with the two experimental data sets. For each algorithm, we used the same sequence of 200 frames. In the first 50 and last 50 frames, the renderings are performed with different image scales. In the middle 100 frames, the renderings are performed using different viewing angles. The desired rendering speed was set to ten frames per second. Figure 2 and 3 (a) show the results of no LOD selection. Based on the texture memory size of our test machine, 33 subvolumes were generated for the delta wing data set, and 39 subvolumes were generated for the brain data set. When the subvolumes were rendered using the highest



(a) no LOD selection (b) octree rendering

(c) feedback algorithm (d) our algorithm

Figure 2: Rendering performance comparison of the four algorithms for the delta wing data set when the desired frame rate is set to 10 frames per second. (a)No LOD selection; (b) Octree based volume rendering algorithm; (c) Feedback control algorithm; (d) Our time critical algorithm.



(a) no LOD selection (b) octree rendering

(c) feedback algorithm (d) our algorithm

Figure 3: Rendering performance comparison of the four algorithms for the brain data set. The desired frame rate is set to 10 frames per second.(a) no LOD selection; (b) Octree based volume rendering algorithm (c) Feedback control algorithm; (d) Our time critical algorithm.

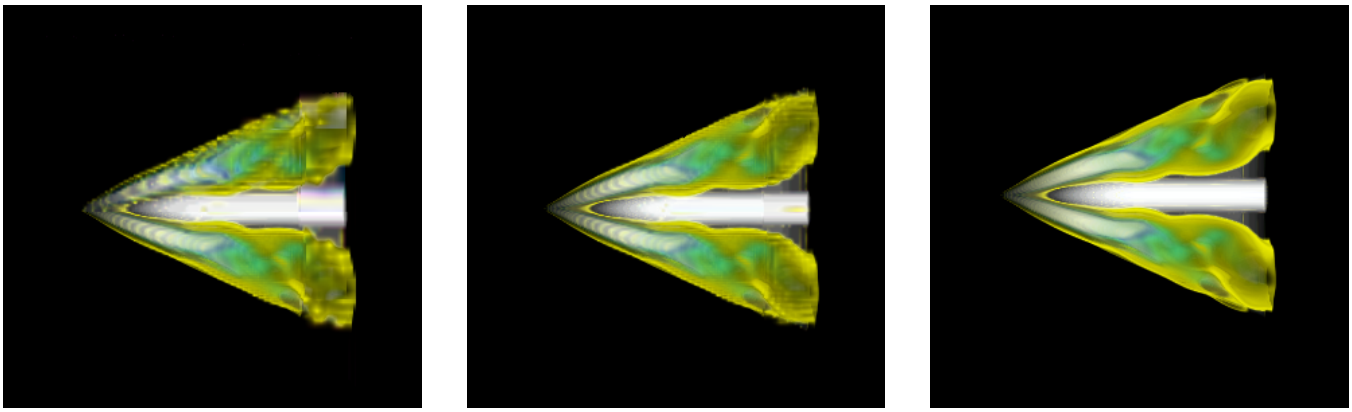
resolution, the rendering speed can not reach the target 10 frames per second. Figure 2 and 3 (b) show the results of the octree based volume rendering in which one fixed error tolerance was used for traversal and rendering. From the figure we can see, by appropriately choosing an error tolerance, we can reach the target frame rate when we only rotated the volume. However, as soon as we zoom in and zoom out to change the volume scales, fixed error tolerance can not guarantee a constant frame rate any longer. Figure 2 and 3 (c) show the results of using the feedback control algorithm, in which the error tolerance is adjusted based on a fuzzy logic control method. The frame rate oscillation is clearly noticeable. Another problem for the feedback control algorithm is that the error tolerance used for adjusting the LOD is not a continuous variable and does not have a direct relationship with the rendering speed. Figure 2 and 3 (d) show the results of the proposed time-critical LOD selection algorithm. Results showed that our control algorithm can guarantee the rendering frame rate within a 10% error range.

5.2 LOD Selection Overhead

In our algorithm, the overhead for selecting appropriate LODs for the subvolumes is very small. Table 1 shows the time spent on the LOD selection algorithm and its relative percentage to the total rendering time for rendering the delta wing data set. Table 2 shows the result for the brain data set. The target frame rate was set to 10, 15, 20, and 25. Results showed that our control algorithm has very small overhead (1-2%) and thus is negligible compared to the total rendering time.

target time (ms)	actual time (ms)	control time (ms)	percentage
0.1	0.0996	0.000998	1.002%
0.067	0.0709	0.001027	1.447%
0.05	0.054	0.000868	1.608%
0.04	0.0402	0.000907	2.257%

Table 1: Control overhead for delta wing data set.



(a) (b) (c)

Figure 4: Comparison between image generated using our time critical algorithm for delta wing data set with and without considering opacity as importance parameter. (a) without considering opacity; (b) considering opacity; (c) full resolution image

target time (ms)	actual time (ms)	control time (ms)	percentage
0.1	0.1048	0.001189	1.135%
0.067	0.069	0.001365	1.976%
0.05	0.0505	0.000922	1.824%
0.04	0.0424	0.000916	2.161%

Table 2: Control overhead for brain data set.

5.3 Flexible Time Budget Allocation

To test the effectiveness of our time budget allocation algorithm, we consider various importance parameters set by the user. For the delta wing data set, the maximum opacity in the subvolume is used as the importance parameter. For the brain data set, a combination of viewing distance and projection area with an equal weight is used as the importance parameter. In the following, we show that our algorithm can successfully allocate more time to render subvolumes that are considered more important. We note that our algorithm can easily incorporate other importance functions as well.

(1) Maximum opacity

Table 3 shows a comparison of time spent to render subvolumes with different opacities using our time-critical algorithm with and without considering opacity importance. The rendering frame rate was set to 10 frames per second. In the table, we categorized the subvolumes into three importance groups based on their maximum opacities - low opacity subvolumes, medium opacity subvolumes, and high opacity subvolumes. From the table we can see, without considering opacity importance, the total rendering time is almost uniformly distributed among the three categories. But with considering opacity importance, high opacity subvolumes were rendered using more time than medium and low opacity subvolumes.

Figure 4 shows images generated using our time-critical algorithm for the delta wing data set. The desired frame rate was set to 10 frames per second. Image(a) was generated without considering opacity importance. Image(b) was generated considering the maximum opacity of each subvolume. Image(c) is the full resolution image with a rendering frame rate of about 6 frames per second. We show that by considering the opacity importance, the image quality was much improved. This is because we allocated more time budget to the areas that are non-transparent.

subvolume category	without considering opacity	considering opacity
low opacity	34.93%	27.74%
medium opacity	32.75%	24.59%
high opacity	32.32%	47.68%

Table 3: Comparison of rendering time distribution for the delta wing data set when using opacity as importance parameter

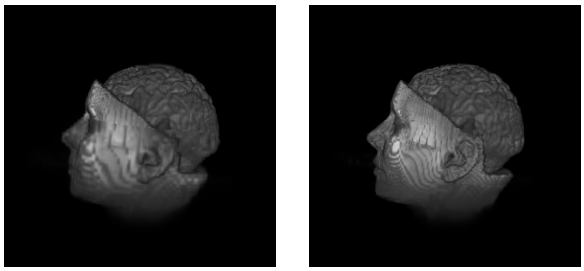
(2) Viewing distance and projection area

The other importance function that we used was a combination of viewing distance and projection area with equal weight. We use this importance function for the brain data set. That is, subvolumes that are closer to the screen and have larger projection area should be rendered in a higher quality as they are less likely to be occluded. To demonstrate the utility of our algorithm, we also categorized the subvolumes into three categories based on their importance values. Table 4 shows the time spent on rendering high, medium and low importance areas using our time critical algorithm, with and without considering the two importance factors. From the table, it can be seen that we are able to increase the time spent on rendering the high importance blocks from 32.29% to 49.59% after taking into account the importance factors in our algorithm.

Figure 5 shows images generated using our time-critical algorithm for the brain data set. The desired frame rate was set to 10 frames per second. Image(a) was generated without considering viewing distance and projection area importance. Image(b) was generated considering the combination of the two importance parameters. We show that by con-

subvolume category	without considering importance	considering importance
low importance	24.98%	31.02%
medium importance	25.43%	36.69%
high importance	32.29%	49.59%

Table 4: Comparison of rendering time distribution for the brain data set when using a combination of viewing distance and projection area as the importance parameter



(a) (b)

Figure 5: Comparison between image generated using our time critical algorithm for brain data set with and without considering viewing distance and projection area as importance parameters. (a) without considering importance factors; (b) considering importance factors;

sidering the importance parameters, the image quality was improved as we allocated more time budget to the areas that are closer to view point and have larger projection area.

5.4 time-critical gaze-directed rendering

In this section, we demonstrate an application of our time-critical algorithm: time-critical gaze-directed rendering. In gaze-directed rendering, areas that are closer to the gaze area are considered to be more important, and therefore should be rendered using high resolutions; areas that are farther away from the gaze area are considered less important, and thus should be rendered using lower resolutions if necessary. To achieve this goal, the gaze distance which is the distance between the gaze center and the center of the projection area of a subvolume can be used as the importance parameter. This way, our algorithm will assign less time to the areas that are farther away from gaze area and save time for areas that are closer to gaze area so that those areas can be rendered at higher quality.

Table 5 and Table 6 show the rendering time distribution with our time-critical algorithm for the two data set with and without considering gaze distance as the importance parameter. The target frame rate was set to 10 frames per second. The subvolumes are divided into three categories based on their gaze distance - small, medium or large. From the two tables we can see, by considering gaze distance as importance parameter, we can effectively allocate more time to render subvolumes that are closer to gaze area. Figure 6 shows the time-critical gaze-directed rendering results for the brain data set when we change gaze area to different parts of the viewport. From the figure we can see that our time-critical algorithm can render areas that are closer to the gaze area with higher resolutions and thus generate better quality images. For the areas that are farther away from the gaze area, as the available time was limited, they were rendered using low resolution and the rendering became blur.

subvolume category	without considering gaze distance	considering gaze distance
large gaze distance	43.39%	12.42%
medium gaze distance	29.26%	24.19%
small gaze distance	27.35%	63.39%

Table 5: Comparison of rendering time distribution for delta wing data set when using gaze distance as importance parameter

subvolume category	without considering gaze distance	considering gaze distance
large gaze distance	20.49%	11.14%
medium gaze distance	33.19%	22.60%
small gaze distance	46.32%	66.26%

Table 6: Comparison of rendering time distribution for brain data set when using gaze distance as importance parameter

6 Conclusion and Future Work

This paper presents a time-critical algorithm for LOD volume selection and rendering using 3D texture mapping hardware. Our algorithm can automatically select appropriate LOD from different regions of the volume to guarantee the desired rendering time, and in the mean time maximize the visualization quality by appropriately distributing the rendering time budget to regions of different importance. Future work includes designing intuitive user interface so that the user can specify importance functions and rendering time budget more easily. We will also experiment different importance functions to achieve different rendering goals.

References

[1] E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of Visualization '99*, pages 355–361. IEEE Computer Society Press, Los Alamitos, CA, 1999.

[2] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3d textures. In *Proceedings of 2000 Symposium on Volume Visualization*, pages 7–13. ACM SIGGRAPH, 2000.

[3] D. Ellsworth, L. Chiang, and H.-W. Shen. Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In *Proceedings of 2000 Symposium on Volume Visualization*. ACM SIGGRAPH, 2000.

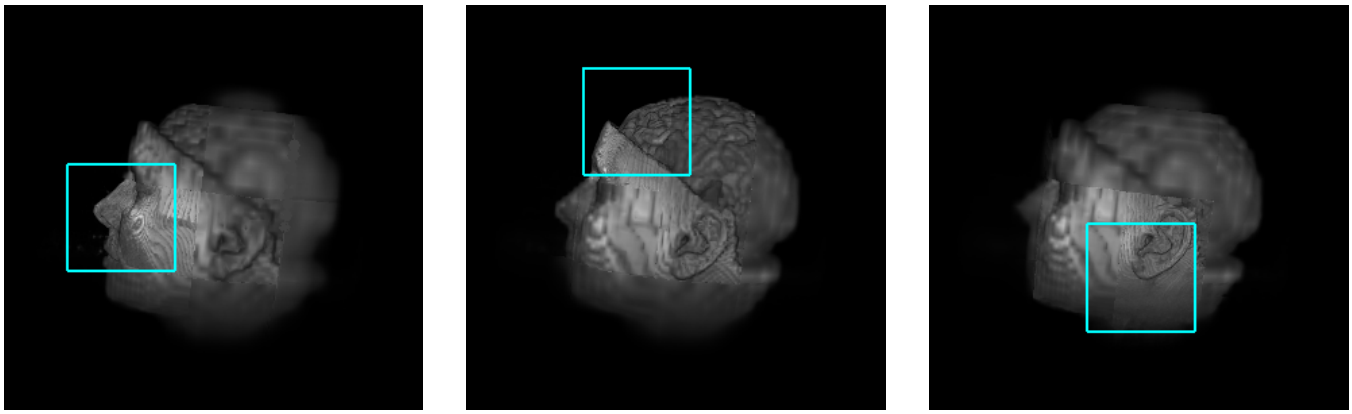
[4] D. Laur and P. Hanrahan. Hierarchical splating: A progressive refinement algorithm for volume rendering. In *Proceedings of SIGGRAPH 91*, pages 285–287. ACM SIGGRAPH, 1991.

[5] T. Funkhouser and C. Sequin. Adaptive display algorithms for interactive frame rate during visualization of virtual environment. In *Proceedings of SIGGRAPH 93*, pages 247–254. ACM SIGGRAPH, 1993.

[6] E. Gobbetti and E. Bouvier. Time-critical multiresolution scene rendering. In *Proceedings of 1999 Symposium on Volume Visualization*, pages 123–130. ACM SIGGRAPH, 1999.

[7] J. Bryan, D. Stredney, R. Wiet, and D. Sessanna. Virtual temporal bone dissection: A case study. In *Proceedings of IEEE Visualization 2001*. IEEE Computer Society Press, Los Alamitos, CA, 2001.

[8] L. Hong, S. Muraki, A Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. In *Proceedings of SIGGRAPH 97*, pages 27–34. ACM SIGGRAPH, 1997.



(a) (b) (c)

Figure 6: Time-critical gaze-directed rendering. The target frame rate is set to 10 frames per second. (a)(b)(c) show the results when gaze area is placed at different parts of the viewport.

[9] M. Reddy, A. B. Watson, N. Walker, and F. L. Hodges. Managing level of detail in virtual environments: a perceptual framework. In *Presence: Teleoperators and Virtual Environments*, pages 658–666, 1997.

[10] Vrml 97, international specification iso/iec is 14772-1. 1997.

[11] *Open Inventor C++ Reference Manual: The official Reference Document for Open Systems.* Open Inventor Architecture Group. Addison-Wesley, Reading, MA, USA, 1994.

[12] X. Li and H.-W. Shen. Adaptive volume rendering using fuzzy logic control. In *Proceedings of Joint Eurographics-IEEE TCVG Symposium on Visualization.* Springer-Verlag, 2001.

[13] J. Rohlf and J. Helman. Iris performer: A high performance multiprocessing toolkit for real-time 3d graphics. In *Proceedings of SIGGRAPH 94*, pages 381–395. ACM SIGGRAPH, 1994.

[14] J. T. Cullip and U. Neumann. Accelerating volume reconstruction with 3d texutre hardware. In *Tech. Rep. TR93-027*, 1993.

[15] M. Woo, Neider J., and Davis T. *OpenGL Programming Guide.* OpenGL Architecture Review Board. Addison-Wesley, USA, July, 1997.