Adaptive Volume Rendering using Fuzzy Logic Control

Xinyue Li and Han-Wei Shen

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

Abstract. This paper presents an automatic error tolerance specification system to control the performance of hierarchical volume rendering. Rather than requiring the user to provide an explicit error tolerance numerically, we let the user to specify only the target rendering speed. Our system can then calculate an appropriate error tolerance adaptively to satisfy the user's performance goal. The system is realized using fuzzy logic control, which enables run-time adaptation based on iterative feedback control and knowledge acquired from past experience. We describe the process of constructing the fuzzy logic control system, and show that the system can successfully steer the performance of volume rendering.

1 Introduction

Direct volume rendering is an effective technique for analyzing three dimensional scalar data, as it allows the user to visualize the underlying field's global structures without the need to generate intermediate geometry. While effective, volume rendering is computationally intensive, which makes interactive manipulation and display of large-scale volume data difficult. In the past, researchers have proposed various software and hardware solutions to accelerate volume rendering. Among these methods, the use of hierarchical data structures and rendering algorithms has proven to be effective as it permits run-time tradeoffs of image quality and rendering speed. In general, this tradeoff is controlled by an error tolerance, which is used to select an appropriate level of details from a multi-resolution volume hierarchy. To generate a rendering result, the volume hierarchy is traversed. If low resolution subvolumes at particular levels of the hierarchy have error measurements smaller than the error tolerance, the subvolumes are used for rendering. Otherwise, the volume hierarchy traversal is continued until all the subvolumes with lower errors are identified.

In general, the error tolerance is specified by the user at run time based on the desired rendering speed and visualization quality. However, as the error measurements for the volume hierarchy are both data and error metric dependent, the rendering speed and image quality corresponding to a particular error tolerance are difficult to predict without extensive knowledge of the underlying data. In addition, the dependency of the volume rendering performance on the image size and transfer function further complicates the process of error tolerance specification. Moreover, in an interactive visualization session where the user's requirements in image quality and computation speed change frequently, it is impractical to assume the user can constantly keep track of an appropriate error tolerance in a time-critical manner.

This paper presents an automatic error tolerance specification system for volume rendering. Instead of specifying a numerical error tolerance which requires a good understanding of the error metric and the underlying data, the user only needs to provide the desired performance in terms of frame rate. Our system is able to locate an appropriate error tolerance automatically to satisfy the desired performance goal. In addition, our system can adjust the error tolerance on the fly when the user varies the viewing parameters. We develop a *Fuzzy Logic Control* algorithm, which allows the system to adapt based on iterative feedback control and the knowledge acquired from past experience. In the following, we first overview related work on hierarchical volume rendering. We then provide a brief overview of fuzzy logic control fundamentals. Details about our adaptive rendering system are provided in section 4, and experimental results are discussed in section 5.

2 Related Work

Researchers have proposed various hierarchical methods to accelerate volume rendering. Levoy [1] used a pyramid data structure to record the volume's local occupancy information based on a binary classification of the voxels, which allows space-leaping and adaptive termination of ray tracing. Laur and Hanrahan proposed a *hierarchical splatting* method, where a pyramid data structure is used and each node of the pyramid stores the corresponding volume block's mean and standard deviation. At run time, a user-supplied error tolerance is provided and an octree is fit to the pyramid. Based on the local error measurements, different regions of the volume can be rendered in different resolutions. Danskin and Hanrahan [2] proposed an *importance sampling* method, where the sampling rate along a ray is changed according to the local and accumulated opacities. A special type of pyramid, called Range$_{27}$ pyramid, is used to facilitate the adaptive sampling. Wilhelms and Van Gelder proposed a selective octree traversal method for hierarchical volume rendering and compression with an extension of storing voxel and cell trilinear functions at the octree nodes [3].

More recently, researchers have proposed to use hierarchical data representations to assist hardware-based volume rendering via 3D texture mapping. The main motivation of the research is to overcome the limitation of texture memory capacity imposed by the underlying hardware. LaMar *et al.*[4] proposed to use a pyramid data structure to construct a texture hierarchy. The leaf nodes of the hierarchy constitute the original volume data and internal nodes store down-sampled volume bricks. The run-time selections of volume bricks are guided by the distances of the bricks to the view point and the center of focus, and their projected angles. Weiler *et al.*[5] proposed a level-of-detail rendering method using a similar data structure with a guarantee of consistent interpolation between different resolution levels. Finally, Ellsworth *et al.*use a Time-Space Partitioning Tree [6, 7] with OpenGL texture object optimization and fast color metric approximations to accelerate time-varying volume rendering.

While all these techniques allow user control to trade image quality for speed, the relations between the control parameters and the rendering speed are not known. As a result, the user often needs to carry out the adaptation on a trial-and-error basis. In this paper, we propose to use a fuzzy logic control method to achieve automatic error tolerance specification for adaptive volume rendering.

3 Fuzzy Logic Control

Unlike the conventional control methods, whose effectiveness strongly relies on the accuracy of the analytic control model, fuzzy logic control is used when developing such a model is difficult or impossible. Like other control mechanisms, fuzzy logic control is essentially a feedback control system as presented in Figure 1. The object to be controlled is called the *system*, denoted as S, which is the volume rendering algorithm in our case. The controller, denoted as C, is to guarantee a desired response of the output y, i.e., keeping the output y close to the reference point w (keeping e small). In our application, y and w are the current and desired frame rates. The output u of the controller C, is the control action, which is the error tolerance in our application. In essence, fuzzy logic control relies on a set of IF...THEN inference rules [8], which have the general form:

if x is A then y is B

where x is the input variable, and y is the output variable. The values A and B are expressed linguistically rather than in numerical forms. Examples of linguistical values are *very low*, *low*, *medium*, *high*, and *very high*. Using the concept of fuzzy sets proposed by L.A. Zadeh in 1965 [9], these linguistic values can be translated into numerical values to perform calculations. The concept of fuzzy set is considered as a generalization of the classical crisp set, which allows objects to take

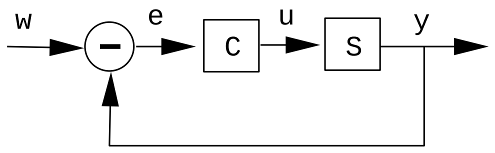

Fig. 1. A feedback control system

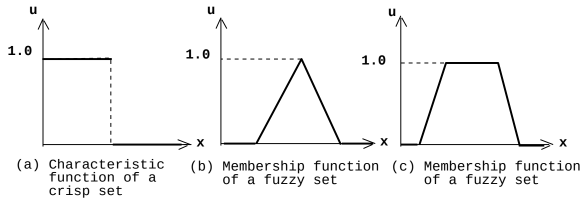
u
1.0
x
(a) Characteristic function of a crisp set
u
1.0
x
(b) Membership function of a fuzzy set
u
1.0
x
(c) Membership function of a fuzzy set

Fig. 2. Crisp and fuzzy sets

partial membership in vague concepts (ie. fuzzy sets). For a crisp set, the degree of an object belongs to the set is either 0 or 1, this relationship can be denoted using a characteristic function as shown in figure 2 (a). For a fuzzy set, however, the degree that an object belongs to the fuzzy set is often a real number between 0 and 1, which is called the membership value in the set. This is denoted using membership functions, as shown in figure 2 (b) and (c).

To build a fuzzy logic control system, a *knowledge base* needs to be first constructed. The knowledge base is composed of two parts, data base and rule base. Data base includes all the membership functions of inputs and outputs of the fuzzy logic control system, which are used to denote the relationship between an accurate value and the linguistic sets. The rule base is used to store all the inference rules of the fuzzy control system, with each rule has the general IF ... THEN ... form. Given the knowledge base which contains both the data base and rule base, fuzzy logic control system at run time will continuously monitor the current system state (the input) and perform the following three steps:

(1) fuzzification: This process is to convert the accurate value of input into fuzzy value, i.e. membership values of the fuzzy sets.
(2) fuzzy reasoning: This is to use the fuzzy inputs and the knowledge base to get the fuzzy output of the control system. This is the kernel of fuzzy control.
(3) defuzzification: The output of a fuzzy logic control system is a fuzzy value which cannot be used directly. This fuzzy output value is converted into an accurate control value by the process of defuzzification.

Figure 3 shows the principal components of a fuzzy logic control system.

4 Automatic Error Tolerance Specification for Adaptive Volume Rendering

In octree-based volume rendering, the trade off between image quality and frame rate is controlled by the error tolerance. A high error tolerance can result in a high frame rate and vice versa. Normally, the user is asked to specify the error tolerance at run time. However, when the user does not have sufficient knowledge about the data set, it can become difficult to predict the relationship between an error and the corresponding frame rate. In this case, the user often needs to go through several trials by specifying arbitrary numerical error tolerance values until a desired frame rate, as well as the image quality, are received. In the following, we present our fuzzy logic control method to realize adaptive frame rate control. In our system, the user only needs to specify the desired rendering speed (frame rate) rather than numerical error tolerance values. Based on the

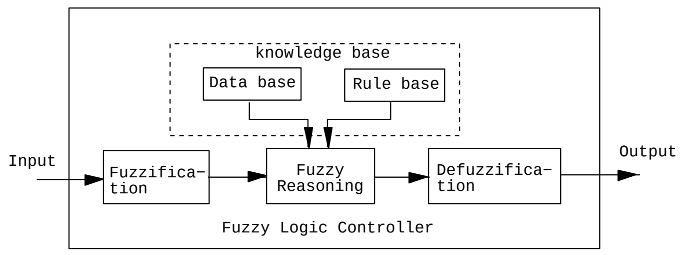

Fig. 3. Principal components of a fuzzy logic control system

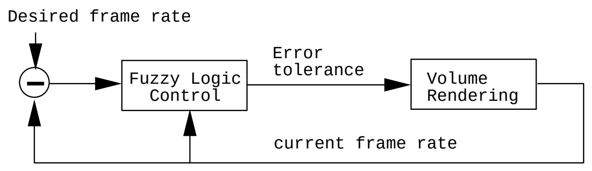

Fig. 4. Adaptive volume rendering system overview

desired frame rate, our system can automatically generate an appropriate error tolerance and the resulting rendering time is approximately at the rate specified by the user. We first briefly overview the hierarchical volume rendering algorithm that we use. We then describe our error specification algorithm in detail.

4.1 Hierarchical volume rendering

The volume hierarchy that we use is similar to the one used by Laur and Hanrahan [10], which is a standard complete octree that recursively subdivides the volume spatially until all subvolumes reach a predefined minimum size. The leaf nodes of the tree point to the raw volume data, and the internal nodes of the tree store the mean values of the corresponding subvolumes. The error metric associated with each internal node is the voxels' standard deviation. At run time, the user specifies an error tolerance, and this error tolerance is compared with the error estimates recorded in the tree nodes. If an internal node's error estimate is smaller than the error tolerance, the mean value is used to represent the corresponding subvolume. Otherwise, the traversal of the octree continues. If a leaf node is reached, the original data in the subvolume are used. To perform the rendering, we use a simple 3D texture hardware acceleration technique proposed by Ellsworth, Chiang, and Shen [7]. This method uses a polygon slicing and 3D texture mapping technique and breaks the volume into small blocks as the result of octree traversal. If an internal node is selected, flat shaded polygons are drawn. Otherwise, slicing polygons within the volume blocks are rendered and mapped with 3D textures. We note that our error specification algorithm described in the following does not use any specific properties of this volume rendering method. Therefore, the algorithm can be applied to other hierarchical volume rendering methods as well.

4.2 Error Specification System Outline

Figure 4 shows an overview of our run-time adaptive volume rendering system. It's a feed back system and the main part of the system is the control unit. The input of the control unit is the current frame rate, and the difference between the user desired frame rate and the current frame rate. The output of the control system is an error tolerance used for volume rendering. In essence, the system works as follows:

(1) After the user specifies a frame rate, the control system randomly chooses an initial error which the rendering system uses to generate an image. The frame rate based on this initial error is then fed back to the input of the control system.

(2) Based on the current frame rate and the difference between the current frame rate and the desired frame rate, the control system adjusts the error tolerance. The basic idea is: ”When the difference is positive, which means current frame rate is higher than the desired frame rate, we need to decrease the error tolerance”. The higher the difference, the larger the error tolerance will drop and vice versa.

(3) The new error tolerance will be used by the volume rendering system again and will result in a new frame rate. If the difference between this new frame rate and the desired frame rate is less than a minimum range, the control system will stop adjusting the error tolerance. Otherwise, this new frame rate is fed back to the input of the control system and repeat step 2 and 3.

Fuzzy Logic Control (FLC) is chosen to implement the above control system. We choose fuzzy logic control for two reasons. First, the input of the control system is a fuzzy variable. There is no explicit threshold to decide that the difference of frame rates is large or small. For instance, if we decide that the value that distinguishes small difference and large difference is 0.8, traditional crisp logic will classify a difference of 0.79 as small, and a difference of 0.81 as large, which is not a desirable decision. Fuzzy logic can solve the problem better as it uses multivalued logic to model problems. For example, instead of saying that a frame rate difference of 0.8 is ”small”, fuzzy logic would state that ” A frame rate difference of 0.8 is 40% belonging to small and 60% belonging to large”. The second reason for using fuzzy logic control is that it is difficult to find a mathematical model to describe the relationship between the frame rate and the error tolerance due to the combination effect of 3D projection, transfer function, visibility, and so on in volume rendering. Fuzzy logic has proven to be quite effective to model such a complex and perhaps nonlinear system.

4.3 The Fuzzy Logic Control System

In this section, our fuzzy logic control system is described in detail. We first present the construction of its knowledge base, and then describe the three main steps, *fuzzification*, *fuzzy reasoning*, and *defuzzification*, for making run time control decisions.

(1) Construction of Knowledge Base As mentioned before, the knowledge base in a fuzzy logic control system consists of a data base and a rule base. The data base includes the membership functions of inputs and outputs, and the rule base contains the inference rules. Our fuzzy control system has two inputs and one output. The input variables are the current frame rate and the frame rate difference (current frame rate minus desired frame rate). The output variable is an error tolerance modulator, which is used to adjust the error tolerance. We need to create one set of membership functions for each input/output variable.
(1.a) Membership functions of input variables: To define the membership function of the frame rate difference, we divide the range of the frame rate difference into five linguistic sets, Negative Big (NB), Negative Small (NS), Zero(ZO), Positive Small (PS), and Positive Big (PB), and the membership functions are shown in figure 5. In the figure, the horizontal axis indicates the difference between the current and the user specified frame rates. The scale of the horizontal axis (frame rate difference) can be scaled differently based on the rendering method and the input data. The vertical axis denotes the membership value of a given frame rate difference. This value is used to indicate the degree to which a difference belongs to a linguistic set. For example, from the figure we can see that the degree of frame rate difference of 0.8 to PS is 0.6, and to ZO is 0.4. In this example, a precise frame difference value is mapped to two linguistic sets.

To generate a new error tolerance, the current frame rate also needs to be taken into account. This is because given the same frame rate difference, the amount of error modulation needed will be different in different frame rate ranges. In our system, current frame rate is divided into three categories, ”High (H)”, ”Medium (M)”, and ”Low (L)”, and the membership functions are shown in figure 6.
(1.b) Membership functions of output variables: We adjust the error tolerance by using an error tolerance modulator M, which is the output of the control system. If the current error tolerance

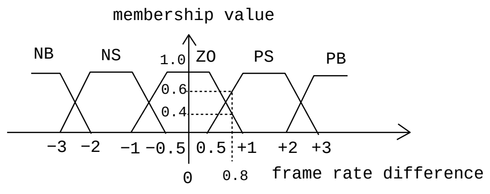

Fig. 5. Membership functions for the input: frame rate difference

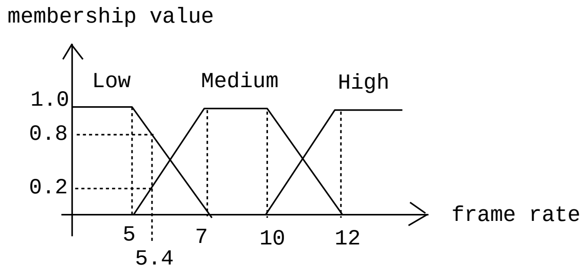

Fig. 6. Membership functions for the input: current frame rate

is E, and the control system output is M, then the new error tolerance becomes $E + E \times M$. To define the membership function of the output variable, the values of the modulator is categorized into fifteen linguistic sets, LNB, LNS, LZO, LPS, LPB, MNB, MNS, MZO, MPS, MPB, HNB, HNS, HZO, HPS, and HPB. We use fifteen output linguistic sets because there are three levels of current frame rate (L, M, H), and five levels of frame rate difference (NB, NS, ZO, PS, PB), which results in fifteen different combinations. We use a singleton membership function to represent each of the linguistic set, as it is easy to perform defuzzification[11]. Figure 7 shows five output singleton membership functions.

To determine the singleton values for the output linguistic sets, we need to perform profiling runs. In essence, the purpose of profiling is to obtain knowledge about the approximate error tolerance modulation needed in order to achieve the desired frame rate. Table 1 shows sample information received from experiments using a sample dataset when the desired frame rate drop is one. Column 1 shows that the current frame rate is categorized into three levels (H, M, L), column 2 shows the initial and target frame rates, column 3 shows the error tolerances used to generate the initial and target frame rates in our profiling with a fixed view, and column 4 shows the corresponding error modulation values. In this example, the frame rate difference (-1) is considered to be “Negative Small (NS)”. To obtain the singleton values for the output linguistic sets HNS, MNS, and LNS, we average the error tolerance modulation values in column 4 for each of the three frame rate categories (High, Medium, and Low). Other singleton values are obtained in the same way. We note that although we use a sample view to get the error tolerance-frame rate information in the profiling runs, the information is sufficient for us to construct a sound fuzzy control system.

(1.c) Creation of rule base: In this step, we construct the fuzzy reasoning rules that govern the relations between the input and output variables. As our present system has two inputs and one output, the form of each rule is: “ IF current frame rate is A and frame rate difference is B, then the error modulation is C”, where A is chosen from “L”, “M”, “H”, B is chosen from “NB”, “NS”, “ZO”, “PS”, “PB”, C is a singleton output. A sample rule can be written as:

If the current frame rate is L, the frame rate difference is PB, then the error tolerance modulator is LPB. (decrease the error tolerance to a high degree)

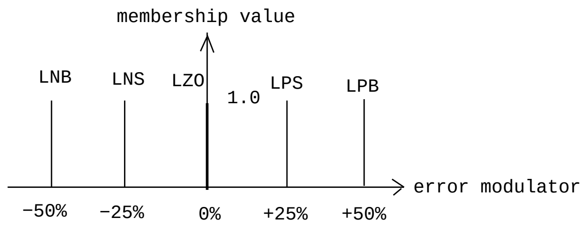

Fig. 7. Output singleton membership functions for error modulator

frame rate	frame rate change	error tolerance change	modulation
High	11 to 10	0.06 to 0.04	-33%
	10 to 9	0.04 to 0.02	-50%
	9 to 8	0.02 to 0.008	-60%
Medium	8 to 7	0.008 to 0.004	-50%
	7 to 6	0.006 to 0.0025	-58%
	7 to 6	0.006 to 0.0025	-58%
	6 to 5	0.0028 to 0.0015	-40%
Low	5 to 4	0.0015 to 0.001	-35%
	4 to 3	0.0012 to 0.0005	-58%

Table 1. Sample profiling results

Table 2 shows the complete rule base. Each entry in the matrix is the linguistic output of the error modulator for the corresponding current frame rate (row) and the frame rate difference (column).

(2) Fuzzification At run time, the accurate values of current frame rate and frame rate difference are converted into fuzzy values based on their membership functions which have been defined in the data base. For example, from the membership functions of frame rate difference shown in figure 5, we can see that a frame rate difference of 0.8 belongs to two fuzzy subsets, PS and ZO, and the membership values are 0.6 and 0.4 respectively, which denote the degrees to which the frame rate difference belongs to the linguistic sets. By this way, an accurate difference value can be mapped to some linguistic subsets and this completes the fuzzification of frame rate difference. The fuzzification of current frame rate can be performed in a similar way.

(3) Fuzzy Reasoning Based on the input frame rate and frame rate difference, fuzzy reasoning will take place to compute an new error tolerance modulator. In essence, there are three steps in fuzzy reasoning, In the following, we use frame rate difference of 0.8 and current frame rate of 5.4 as an example to explain the three steps:

(3.a) Find the firing level of each rule: The firing level of a rule is determined by the satisfaction of each of the component in the antecedent of the rule. From the membership functions shown in figure 5 and figure 6, we can see that a frame rate difference of 0.8 is related to two linguistic sets, PS and ZO, and the membership values are 0.6 and 0.4 respectively; the current frame rate of 5.4 is related to two linguistic sets, L and M, and the membership values are 0.8 and 0.2 respectively. As a result, four rules, rules LPS, LZO, MPS, MZO, shown in table 2 will be activated. The firing level of a rule is calculated by combining the satisfaction of each of the antecedent component. Normally we use Min aggregation for combination. For instance, in the case that the degree of ”current frame rate” in L is 0.8 and ”frame rate difference” in PS is 0.6, then the firing level for the rule LPS is Min(0.8, 0.6) = 0.6.

(3.b) Find the output of each rule: Looking at table 2, we can know that the output for rule LPS is “the error tolerance modulator is LPS”, for rule LZO the output is “the error tolerance modulator is LZO”, and so on.

FRD Output CFR	NB	NS	ZO	PS	PB
L	LNB	LNS	LZO	LPS	LPB
M	MNB	MNS	MZO	MPS	MPB
H	HNB	HNS	HZO	HPS	HPB

FRD: Frame Rate Difference
Output: Error Modulator Output
CFR: Current Frame Rate

Table 2. Fuzzy reasoning rule base

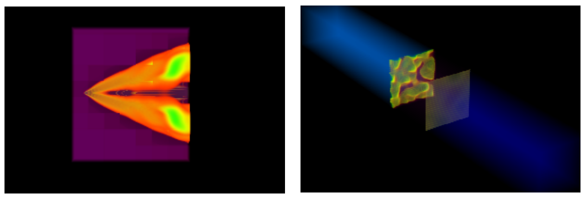

Fig. 8. Images for the delta wing and shockwave data sets

(3.c) Aggregate the individual rule: Based on the activated rules and the firing level of each rule, the overall output of the control system is an aggregation of the output from step 1 and 2. In our example, the degree of output error tolerance modulator belongs to LPS is 0.6, to LZO is 0.4, to MPS is 0.2, to MZO is 0.2. Note that at this stage, the system output is still a fuzzy value, we need to have a defuzzification step as described in the next step to get a usable numerical output for the error tolerance modulator.

(4) Defuzzification The defuzzification process is needed for calculating a numerical output from the aggregation of individual fuzzy output. In our case, it is to determine the actual value of the error tolerance modulator. We use center of area (COA)method [12] to perform defuzzification. That is, given the firing level F_i for the rule(i), and the rule's corresponding singleton output S_i, the defuzzified value is:

$$M = \sum(F_i \times S_i) / \sum F_i$$

For instance, if the singleton outputs for LPS, LZO, MPS, and MZO are -25%, 0%, -10%, and 0%, and the firing levels for the rules are 0.6, 0.4, 0.2, 0.2, then the final error tolerance modulator is:

$$M = (0.6 * -25\% + 0.4 * 0\% + 0.2 * -10\% + 0.2 * 0\%)/(0.6 + 0.4 + 0.2 + 0.2)$$

As a result, the new error tolerance is calculated as: new error tolerance = current error tolerance $\times(1 + M)$.

5 Results and Discussion

We have implemented our automatic error specification algorithm with a hardware based volume rendering method mentioned before. All the experiments shown in this section were performed

delta wing			shockwave		
desired	result frame rate	iterations	desired	result frame rate	iterations
4	4.27	5	4	3.78	6
5	4.52	4	5	5.27	7
6	6.24	4	6	5.97	7
7	7.44	4	7	7.19	7
8	8.03	4	8	7.84	8
9	9.12	6	9	9.03	6
10	10.42	4	10	10.46	5

Table 3. The desired frame rate, result frame rates, and iterations

on an SGI Octane with one MIPS R12000 processor, 512 Mbytes memory, and 4Mbytes texture memory. Two regular Cartesian gridded datasets were used in our tests. One is a delta wing data set with resolution $111 \times 126 \times 51$, and the other is a shockwave data set with resolution $512 \times 64 \times 64$. Figure 8 shows the sample images for both data sets. We tested the effectiveness of our fuzzy logic control system under two scenarios. One is to let the user specify an initial frame rate requirement when the system just starts up, and the other is to allow the user to incrementally change the frame rate requirement on the fly. In the first scenario, our error specification system initially chose a random error tolerance and then performed the feed back control to adjust the output. Our goal is to have the system converge to the desired frame rate with a minimum number of iterations. Table 3 shows the results for both the delta wing and shockwave data sets.

From the results, we can see that even with a randomly picked error tolerance in the beginning, our automatic error specification system can effectively identify appropriate error tolerances after a small number of iterations. Table 4 shows the results when the user incrementally changed the desired frame rate dynamically for both the delta wing and shockwave data sets. Our fuzzy control system can adjust the error tolerance efficiently.

delta wing				shockwave			
initial	target	result frame rate	iterations	initial	target	result frame rate	iterations
12	10	10.20	2	8	6	5.6	3
10	8	8.13	1	7	5	4.74	1
8	6	6,.28	1	6	4	3.6	1
6	4	4.39	1	5	3	3.42	1
4	6	5.87	1	4	2	2.27	1
6	8	8.30	1	2	4	4.22	1
8	10	9.75	2	4	6	6.39	3
10	12	11.82	2	6	8	7.70	1

Table 4. The initial frame rate, target frame rate, result frame rate, and iterations when the user incrementally changed the desired frame rate

Our fuzzy logic control system constantly monitors the rendering performance at run time and adjusts the error tolerance when necessary. For instance, when the user scales the volume data to different sizes, the rendering time will change since the speed of most of the volume rendering algorithms are sensitive to the projection area of the volume. Table 5 shows the results from the delta wing and shockwave data sets when the user dynamically changed the scale of the object and the desired rendering speed was ten frames per second. Five different scales resulted in five different error tolerances, and therefore five different levels of details in the rendering results. Our

	delta wing			shockwave		
scale	result frame rate	error tolerance	iterations	result frame rate	error tolerance	iterations
scale 1	10.42	0.022	1	9.58	0.821	1
scale 2	10.04	0.0098	1	9.94	0.469	1
scale 3	9.61	0.0067	1	9.52	0.445	1
scale 4	9.68	0.0056	1	9.7	0.359	1
scale 5	9.62	0.0031	1	10.11	0.397	1

Table 5. The actual frame rate, the error tolerance, and the number of iterations when the user interactively change the scale of the volumes

automatic error specification algorithm was able to maintain the frame rate close to the user's expectation.

6 Conclusions and Future Work

We have presented an automatic error tolerance specification system for hierarchical volume rendering using fuzzy logic control. Our system can dynamically track the performance of the rendering program and calculate appropriate error tolerances to satisfy the user's performance goal. Our system allows the user to control the performance tradeoff more intuitively without the need to have extensive knowledge about the data. We have showed that fuzzy logic control has a great potential to assist in constructing adaptive volume rendering algorithms.

Future work includes integrating the control system with a variety of volume rendering methods and applications. We will also explore the use of fuzzy logic control to precisely maintain the image quality. Furthermore, we intend to integrate the control system with operating system support to create a resource aware adaptive visualization framework.

References

1. M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.
2. J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 91–99. ACM SIGGRAPH, 1992.
3. J. Wilhelms and A. Van Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 27–34. ACM SIGGRAPH, 1994.
4. 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.
5. 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.
6. H.-W. Shen, L.J. Chiang, and K.L. Ma. A fast volume rendering algorithm for time-varying field using a time-space partitioning (tsp) tree. In *Proceedings of Visualization '99*. IEEE Computer Society Press, Los Alamitos, CA, 1999.
7. 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.
8. L.A. Zadeh. The calculus of fuzzy if/then rules. *AI Expert*, 7:23–27, March 1992.
9. L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
10. 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.
11. K. Tanaka. *An introduction to fuzzy logic control for practical applications*. Springer New York, 1997.
12. R. Yager and D. Filev. *Essentials of fuzzy modeling and control*. J. Wiley New York, 1994.