Fall 2010

Error-driven adaptive resolutions for large scientific data sets

Samuel H. Vohr
University of New Hampshire, Durham

ERROR-DRIVEN ADAPTIVE RESOLUTIONS FOR LARGE SCIENTIFIC DATA SETS

BY

Samuel H. Vohr

B.S., University of New Hampshire (2007)

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science

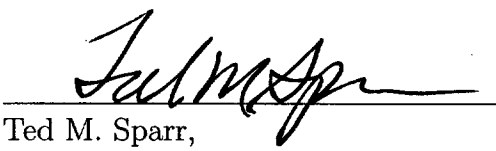
in

Computer Science

September 2010

This thesis has been examined and approved.

Thesis director, R. Daniel Bergeron,
Professor of Computer Science



Ted M. Sparr,
Professor of Computer Science

Philip J. Hatcher,
Professor of Computer Science

8/9/2010

Date

ACKNOWLEDGMENTS

I would like to thank my co-advisors, R. Daniel Bergeron and Ted M. Sparr, for their help and guidance through the entire project and Philip J. Hatcher for his advice as a member of my committee. I would also like to thank Andrew Foulks for his expertise and helpful suggestions along the way and Jimmy Raeder and the OpenGGCM Group at the UNH Space Science Center for the use of their data set. Finally, I would like to thank Ethan Burns and Jordan Thayer from the UNH Artificial Intelligence Group for the advance use of their plotting tool, SPT.

This work was supported in part by NASA under grant AISR05-0071, and the National Science Foundation under grant IIS-0082577.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

ERROR-DRIVEN ADAPTIVE RESOLUTIONS FOR LARGE SCIENTIFIC DATA SETS

by

Samuel H. Vohr
University of New Hampshire, September, 2010

The process of making observations and drawing conclusions from large data sets is an essential part of modern scientific research. However, the size of these data sets can easily exceed the available resources of a typical workstation, making visualization and analysis a formidable challenge. Many solutions, including multiresolution and adaptive resolution representations, have been proposed and implemented to address these problems.

This thesis describes an error model for calculating and representing localized error from data reduction and a process for constructing error-driven adaptive resolutions from this data, allowing fully-renderable error-driven adaptive resolutions to be constructed from a single, high-resolution data set. We evaluated the performance of these adaptive resolutions generated with various parameters compared to the original data set. We found that adaptive resolutions generated with reasonable subdomain sizes and error tolerances show improved performance during visualization.

CHAPTER 1

INTRODUCTION

Modern scientific research often involves visualizing data sets whose size greatly exceeds the resources available on a typical, commodity workstation. As a direct result, making observations and drawing conclusions from large scientific data sets remains challenging for researchers. Interactively displaying an entire multi-gigabyte or terabyte data set is impractical due to several bottlenecks, the most limiting of these being the cost of I/O operations. Fortunately, interesting behavior can be recognized through wide-range, low resolution views and narrower-range, higher resolution views.

A *multiresolution data model* is a popular method for addressing these considerations. Several lower resolution data sets are generated by applying a data decomposition algorithm to the original data set. With each successive lower resolution, the data sets become smaller and, consequently, more easily manipulated in interactive time. However, with each lower resolution, more and more detail is lost from the original data set. To reduce the impact of the lost data, the loss can be calculated and presented to the user or application for consideration along with the low resolution data set. In this work, we present an *error model* for generating and representing the *uncertainty* associated with low resolution data.

Using this model, researchers are afforded several key capabilities. The low resolution data provides the scientist with *context*, the errors highlight *regions of interest*, and the high resolution subsets give the scientist *focus* on regions of interesting subsets of data, without losing the quality of the data. Combining these three functions can result in interactive renderings of dynamic time series data sets that cannot fit into main memory.

Error information can also be used by other applications to account for uncertainty in their computations. One such application is the generation of *error-driven adaptive resolution* representations. Using adaptive resolution, subvolumes are independently assigned resolutions to capture detail for regions of interest, while presenting coarse overviews for less important areas of the domain. In error-driven adaptive resolution, subvolumes are assigned resolutions based on their error values and an error tolerance specified by the user. Regions with low error values can be safely reduced in resolution while regions with high amounts of change are preserved at higher resolutions. The user is presented with a representation of the domain that meets an acceptable error tolerance, while reducing the overall I/O required for the representation.

When used in combination, these tools allow the user to go from a large, high resolution data set to a multiresolution data set with error data to a fully-renderable, error-driven adaptive resolution representation. This provides scientists a range of options for finding the best representation for exploring their data.

1.1 Thesis Goals

The goals of this thesis are:

- Define an error model for representing localized error (by point or by subdomain) from data reduction.
- Develop a tool based on the model for generating multiple low resolutions with error information from a single data set.
- Develop a tool to construct complete, error-driven adaptive resolutions from the multiresolution data sets.
- Evaluate the performance of these representations based on both I/O and computational criteria.

1.2 Outline

- Chapter 2 of this thesis presents some of the prior work on which this thesis is based as well as some similar research done by others.
- Chapter 3 presents a flexible error model for representing localized error caused by data reduction from which error-driven adaptive resolutions can be generated.
- Chapter 4 describes the process for constructing adaptive resolutions from a multiresolution data set with localized error.
- Chapter 5 describes the tests and data sets used in evaluating our error-driven adaptive resolution representations in comparison to the original, high-resolution data sets. The results from these tests are also presented.
- Chapter 6 of this thesis discusses the conclusions we have drawn from the test results and suggests some directions for future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Multiresolution and Adaptive Resolution Data

This work is, in part, an extension of the model for multiresolution and adaptive resolution data developed by Rhodes et al. [21, 20, 22, 19]. This model forms the basis for the GRANITE SCIENTIFIC DATABASE SYSTEM. In the model, the geometry and topology of the data set are conceptually separated, allowing for many different types of data sets to be accessed through the database system. This thesis focuses on constructing *adaptive resolution representations* from *non-adaptive multiresolution hierarchies* using *localized error caused by data reduction.*

Both multiresolution and adaptive resolution have been used before in many applications. Freitag and Loy combine adaptive and multiresolution representations with parallel rendering algorithms[12]. The result is a system that allows large data sets to be explored in interactive time while still retaining high resolution in areas of interest. They hint at error-driven adaptive resolution but do not describe it in detail. The adaptive resolution representations generated by our tool could be incorporated into their distributed octree infrastructure and rendered in parallel.

Adaptive mesh refinement has been used in many applications for quite some time [25, 11]. However, these methods are used at the time of simulation or data collection,

while the model of adaptive resolution described in this proposal can be applied later in the process.

2.2 Quality Assessment

Low resolution representations sacrifice some level of quality for a greatly reduced space requirement. This loss can be mitigated by storing some representation of how the new data set differs from the original. Wang and Ma have developed a statistical approach for quality assessment in volume data [24]. Their method analyzes the features of a data set (either reduced in resolution or otherwise distorted) to calculate a distance measurement from the original data. However, the method they propose yields a single, cumulative value to represent the data set's overall quality. Our visualization model requires a *localized* approach, that yields many values associated with points or regions in the original domain.

2.3 Geometry and Topology

Many variations of computational grids have been used to generate and store scientific data[23, 6, 17]. These range from simple, uniform grids to complex, unstructured grids. Computational grids can be classified by the complexity of the mapping of *index space* coordinates to *geometry* space coordinates. Regular grids (such as the one in Figure 2-1c) are characterized by the condition that any point (i, j, k) in index space is mapped to $(i * dx, j * dy, k * dz)$ where dx, dy and dz are constants. Regular and uniform grids are the simplest to implement, but do not allow for much adaptability. In a rectilinear grid (Figure 2-1d), the distances between points in each row, column or slice are arbitrary, but the cells remain rectangular prisms. Point (i, j, k) in index space is located at $(x[i], y[j], z[k])$ in geometry space where $x[i]$, $y[j]$ and $z[k]$ are table look-ups or some other functions of i, j or k. If these functions take all of the point coordinates into account ($(x[i, j, k], y[i, j, k], z[i, j, k])$), we have a *structured* grid or a *curvilinear* grid. Figure 2-1e is a structured grid where table look-ups provide the positions of each point, allowing for points to be placed arbitrarily

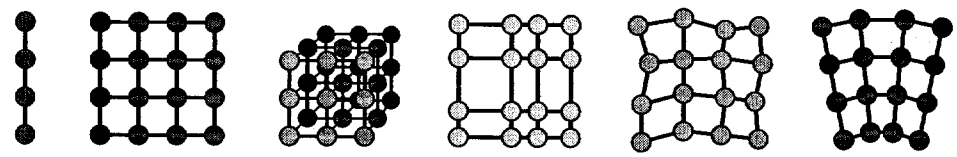


Figure 2-1: Some rectilinear topologies. From left to right: one-dimensional grid, two-dimensional grid, three-dimensional grid, two-dimensional perimeter lattice, non-uniform two-dimensional grid, curvilinear grid.

within the rectilinear topology. Figure 2-1f shows a curvilinear grid where each point's position is located in a non-linear fashion.

All of these grids use a rectilinear topology, but *unstructured grids* can also be used. Unstructured grids allow for the most freedom in geometry and topology, but this comes at the cost of explicitly representing this information for each point.

Although GRANITE supports many different types of dimensional data, this work deals specifically with data sets with *rectilinear* topologies, such as those shown in Figure 2-1. This class of data can be stored in an array in such a way that the neighbors for each point can be found easily through the point's index and dimension information. As a result, the topology information is stored implicitly, avoiding the need to store additional, potentially complex, topology data.

CHAPTER 3

ERROR MODEL

3.1 Introduction

Large data sets can easily be reduced in size through data decomposition methods. However, these reductions come with the cost of lost accuracy in the low resolution version. To offset this, error values can be calculated and stored along with the new data set. These error data sets can be examined by researchers along with the low resolution data through visualization tools, or can be used by other programs to account for error in their computations.

The process for generating data and error sets is straightforward. First a low resolution data set is generated from the original. This can be done by any one of several methods. Next, error values are calculated by comparing values derived from the original and low resolution data sets. Values at points not stored at either resolution are calculated using an *approximation function* (typically some form of interpolation). The result is a set of values that represent the error (or estimated error) for a set of points within the data space. These points can be grouped and summarized by subdomain or *region* to produce localized error values for each region in the domain.

The index space coordinate system for the original data set can be used to describe the placement of both error and data points in all lower resolutions. However, it is convenient to introduce a new coordinate system, the *base resolution*, that shares its origin with the original and has twice the resolution. As a result, data at point (n, m) in the original resolution is represented at point $(2n, 2m)$ in the base resolution. Intermediate points in the original resolution, such as the locations of wavelet summary values in the low resolutions,

can be easily represented with integer values.

For simplicity, the following examples are presented in two dimensions, but can easily be extended to more.

3.1.1 Formal Definitions

The formal definitions of *data representations*, *data models*, and *multiresolution hierarchies* upon which the GRANITE SCIENTIFIC DATABASE SYSTEM is built [19] can be extended to meet the needs of this error model. A brief overview of the definitions used for the error model follows.

Formally, a *data representation* consists of a continuous domain (D), a value space (V), a discrete set of sample points (Δ, such that $\Delta \subset D$), and a function that maps sample points to values (f_Δ) or:

$$R = \langle D, V, \Delta, f_\Delta \rangle$$

A data representation can be paired with an approximation function (f_A) resulting in a *data model*:

$$M = \langle R, f_A \rangle$$

The original data model definition includes components that represent the uncertainty associated with the original sample points (Δ), introduced through the initial measurement or simulation. Alone, these are not sufficient to represent the error associated with low resolution versions of the data set, since error for points not contained in Δ should also be considered. Although important, these items are not within the scope of an error model for representing error as a result of data reductions.

A *reducing operator* (R) can be used to produce a new resolution with a reduced number of sample points ($\mid \Delta \mid$). This results in a lower resolution representation of the same domain, and a *localized reduction error*.

$$\mathsf{R} : M \rightarrow (M', E_r)$$

The localized reduction error component is not defined in detail in the original model. Our error model proposes two possible error representations that could be used for E_r.

A *multiresolution hierarchy* is defined as a pair consisting of a sequence of *levels* (Λ) and a sequence of reducing functions (ρ).

$$\mathcal{M} = \langle \Lambda, \rho \rangle$$

Each level Λ_i represents the result of applying a reducing operator ρ_i to level Λ_{i-1}. This yields a pair containing a data model (M^i) and a *localized reduction error* (E_i).

$$\Lambda_i = \langle M^i, E_i \rangle$$

Λ_0 holds the original data model and its error, which is assumed to be zero.

3.1.2 Localized Error Goals

There are several desirable characteristics for an error model to fill the role of the localized reduction error. 1) The error model must be able to produce an error value for any point in the domain. If an error value for a point is not stored directly, an approximation can be used in its place. 2) In most cases, it is preferable for this approximation to overestimate the error, rather than underestimate. 3) We also want to limit the size of the error representation to be no larger than the size of the low resolution data set it accompanies.

3.2 Data Reduction

Many different techniques exist for generating low resolution data sets. Our error model focuses on techniques that *globally* reduce the resolution of a data set. That is, the data set is reduced substantially with one iteration of the algorithm. Algorithms that incrementally reduce the resolution of a given data set are not considered (mesh decimation, for example [5, 7]). Our preliminary implementation supports two specific techniques, uniform decimation and wavelet decomposition. However, other techniques that produce a global data set reduction could also be used.

Methods used to generate low resolution versions of data sets are formally modeled as a reducing operator (R). This operator performs a transformation on a data model, reducing the number of sample points while preserving the domain D of the original data set.

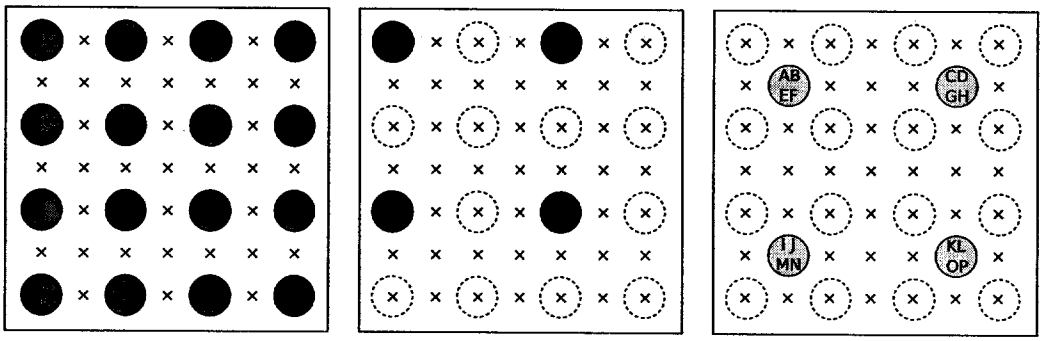
3.2.1 Uniform Decimation

In *uniform decimation*, a lower resolution data set is generated by selecting points from the original at regular intervals [13]. The interval can be any integer but, by default, each interval in a single decimation step is a power of 2. For convenience, the resolutions to be generated are referred to by their exponent. A lower resolution generated by retaining every other data point (an interval of 2) is referred to as the first resolution. One generated with an interval of 4 (2^2) is referred to as the second resolution. The second resolution can also be thought of as the first resolution generated from the first resolution of the original data set. To generalize, any resolution r can be generated by retaining every 2^rth data point in the original resolution or by retaining every other data point in resolution $r-1$.

The data points that make up any lower resolution retain their positions from the original resolution.

3.2.2 Wavelet Decomposition

Wavelet decomposition algorithms can also be used to generate multiple resolutions. These include Haar and Daubechies wavelets [8, 14, 3, 4, 10, 15]. To build a low resolution representation for a one-dimensional data set, an orthogonal wavelet is applied to a discrete data set of size n, resulting in $n/2$ summary values and $n/2$ detail values. In order to reduce the required storage space, the detail information can be discarded or further reduced in resolution, independently from the data resolution. In a multi-dimensional data set, the transform is applied along each dimension. For a two-dimensional data set, a one-dimensional transform is applied along each row, then again along each column of the result. The final result is a summary data set 1/4 the size of the original data. The resulting data set is called the first resolution. Repeating the process on the first resolution data produces



(a) (b) (c)

Figure 3-1: (a) Original data set. (b) First resolution generated by uniform decimation. Sample points are shown as red circles. (c) First resolution generated using wavelet summary values. Summary points are shown as yellow circles.

the second resolution.

Since the data points that make up a lower resolution are summarized from multiple points in the original data set, the position of the new data point should not be the same as a point in the original resolution. Instead the new datum is placed, by default, at the center of all points used to calculate it. In geometry space, this can be done by averaging the points, or applying the same wavelet function to each dimensional component. In index space, the intermediate points in the base resolution can be used. In the two-dimensional example, a value calculated from points $(0,0)$, $(0,2)$, $(2,0)$, and $(2,2)$ in the *base resolution* would be located at $(1,1)$ as shown in Figure 3-1c.

3.3 Approximating Functions

Local error values are calculated by comparing values in the original data set with values in the new, low resolution version. An approximating function f_A can be used to calculate intermediate values not contained in the current data representation. Normally, this approximating function produces the same result for each sample point as the sampling

function or:

$$\forall d \in \Delta, f_A(d) = f_\Delta(d)$$

Bilinear (2D) or trilinear (3D) interpolation is used by default to estimate intermediate values. In addition to being the most general method, it is also the most appropriate for data sets intended to be visualized, as the graphics hardware will also use bilinear or trilinear interpolation when rendering the low resolution data set. However, alternative methods can also be used in its place. In fact, the process of finding a better approximation function can be likened to the formation of a new scientific hypothesis.

3.4 Error

3.4.1 Point Error

Error is generally defined as the difference between an estimated value with its known, correct value [16]. Applied to this model, the error at any given point (including points not contained in the original data set) would be the difference between the value at that point derived from the low resolution set and the value derived from the original data. We can describe the error at any point $p \in D$ as

$$E_r(p) = M^r.f_A(p) - M^0.f_A(p)$$

Where r is the resolution level.

Since an error value can be calculated for any point in the data space, the error for the data set can be considered a continuous distribution over the same space as the original data. Point error sets are constructed by sampling from this distribution, producing a *data representation* for the error data.

$$E_r = \langle D, V_E, \Delta_E, f_{\Delta_E} \rangle$$

This can be paired with an approximating function to estimate unsampled locations. Most

error representations are based on this fundamental comparison of data points from the original set with points from the low resolution.

As with any form of sampling, there are limitations on the quality of any error information reconstructed from the set of sampled points (including especially the Nyquist limit [9]).

3.4.2 Region Error

Calculating error associated with individual points can produce reliable representations of the data set's error. However, insights can also be gained by calculating localized error for regions within the original domain. Values can be calculated for each region to represent the maximum error, mean error with standard deviation, median error, signal-to-noise ratio or some other aspect of the error. The user can pick the error function(s) that best fit the application, as well as the desired error resolution.

To calculate these values, the domain is first partitioned into *error* regions. The size of the new error set depends on the number of regions used. More (smaller) regions result in higher resolution error sets, while fewer (larger) regions produce coarser resolutions. Generally, each dimension of an error region is 2^r ($2^r + 1$ when edges are shared) in size, where r is the desired error resolution level. The region is then sampled to build a list of point error values. By default these sample points are located at the original data points, but they need not be limited to only those. Finally a summary function is applied to the list of error values, producing a single value for the region.

Formally, a region $\mathcal{R}$ is a nonempty subset ($D_{\mathcal{R}}$) of the domain D and a set of points ($\Delta_{\mathcal{R}}$), sampled from the subdomain $D_{\mathcal{R}}$:

$$\mathcal{R} = \langle D_{\mathcal{R}}, V, \Delta_{\mathcal{R}}, f_{\Delta_{\mathcal{R}}} \rangle$$

where $D_{\mathcal{R}} \subseteq D$ and $\Delta_{\mathcal{R}} \subset D_{\mathcal{R}}$. The error for an entire region can be calculated by calculating the point error for every point in $\Delta_{\mathcal{R}}$ and applying a summary function f_S

(maximum-absolute value, average, signal-to-noise, etc).

$$\mathcal{E}_r(\mathcal{R}) = f_S(E_r(p) : \forall p \in \Delta_{\mathcal{R}})$$

A *region error* data set is based on a set of regions (δ) that cover the entire domain D from the original data set. Because regions are allowed to share edges or overlap, it is not a requirement that the intersection of any two regions in δ be empty. A region error data set can be represented as a data representation:

$$\mathcal{E}_r = \langle D, V_E, \delta, f_\delta \rangle$$

In this representation, the set of sample points (Δ) is replaced with a set of regions (δ) and f_Δ is replaced with a function that maps regions to error values (f_δ).

Using region error allows the user to consider all original sample points when calculating error values. Depending on the chosen summary function, this can reduce or increase the influence of an errant spike in the data set. If the user is interested in spikes, a function such as maximum absolute error can be employed, otherwise an average of error values could be used. Region error data sets can be rendered like point error data, by defining a point location for each error value (usually the region's center). Region error is particularly useful in large data sets in which significant variation in error can occur in different regions of the data. One suitable application is the construction of error-driven adaptive resolutions from multiresolution data sets. This is discussed in Chapter 4.

3.4.3 Other Error Metrics

There are potential error metrics that are not based on the comparison of values derived from different resolutions. For example, wavelet detail values could be used to represent the error for the points in the data set. This model supports the use of such metrics.

3.4.4 Error Notation

Since there is the potential within a multiresolution hierarchy to have multiple low resolutions each with multiple error sets associated with it, a convenient error notation is required.

$E_{i,j}$ is used to reference the error set at resolution j associated the data set at resolution i (D_i). A data set can have several error sets associated with it ($E_{i,j}$, $E_{i,k}$, etc.).

CHAPTER 4

ERROR-DRIVEN ADAPTIVE RESOLUTION

Using the original data set, multiple low resolution data sets can be generated, each with multiple sets of error information. From these data sets and an error tolerance specification, an *error-driven adaptive resolution representation* of the full data set can be built. Through adaptive resolution, regions of the data set that demonstrate high amounts of unpredictable[1] change are placed at higher resolutions while regions with less or predictable change, can be represented at lower resolutions. To determine the resolution to be assigned to the region, the region's error values are compared against an error tolerance. In this chapter, we describe the process for building our error-driven adaptive resolutions.

4.1 Components

4.1.1 Data Sets

A data set with multiple resolutions is required to build an adaptive resolution representation. This can be done with data generated through uniform decimation, wavelet decomposition, or some other means.

[1]The reconstruction algorithm produces a value that differs greatly from the actual value.

4.1.2 Error Sets

Error information for each resolution in the multiresolution set is required to drive the adaptive representation. The original resolution is considered to be error free. Any error metric can be used in building adaptive resolutions, provided there is a function to compare error values. Error sets used for building adaptive resolutions can usually be kept at much coarser resolutions than their corresponding data sets.

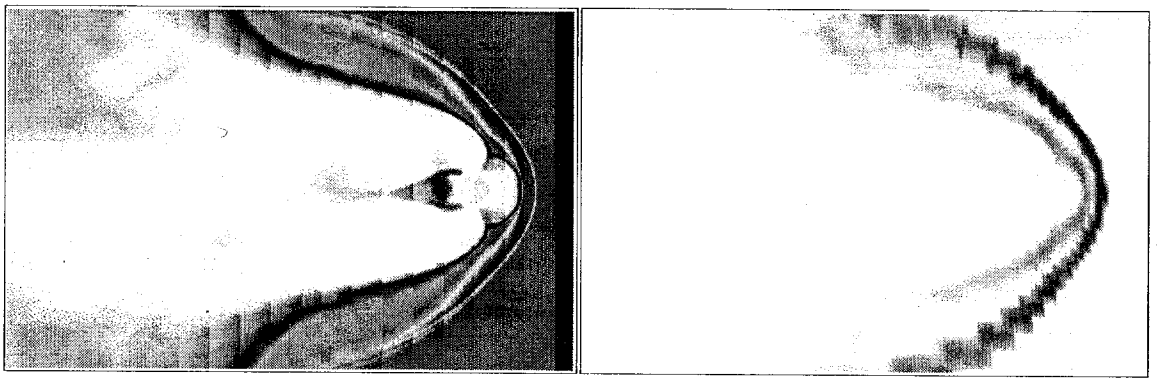
Many different error configurations can be used, but in the following examples, we use a set of error sets in which each level uses the same resolution offset between its data and error sets ($E_{1,3}$, $E_{2,4}$, $E_{3,5}$ for example). This has the advantage that each error point at every level represents the same number of data points.

4.1.3 Subdomains

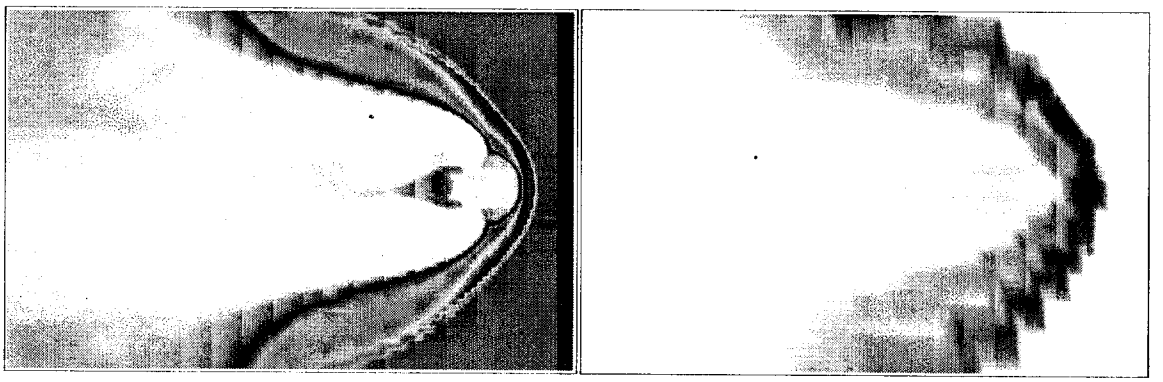
Building an adaptive representation requires the original domain to be divided into subdomains that can be placed at different resolutions. An octree (or quad tree in the case of two dimensions) can be used to organize the subvolumes. Each node in the tree stores a reference to a corresponding volume at some level in the multiresolution data set, along with an error value to represent the quality of the volume's data (see Figure 4-2).

Subvolumes are divided along the boundaries of the regions used in the error set. This avoids the case where an error value is split across two regions and the case where a subvolume is smaller than an error region. For simplicity, we use the error resolution to define the subvolumes in our adaptive resolutions, but it is possible to build representations with more than one error value per subdomain.

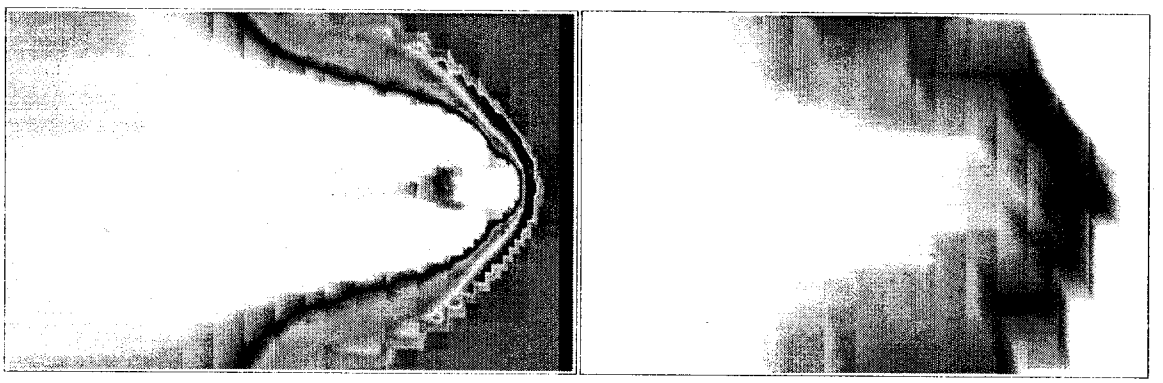
The number of subdomains strongly affects the performance and the effectiveness of the adaptive resolution. Small subdomains can closely fit the features of the data set, reducing the need to use the finest resolutions, but the overhead associated with so many subdomains may negate any benefits. Large subdomains avoid this cost, but risk assigning unnecessarily high resolution levels to certain subvolumes. The size of subdomains is a key parameter



D_1 $E_{1,3}$



D_2 $E_{2,4}$



D_3 $E_{3,5}$

Figure 4-1: Slice renders of three low resolution data sets generated through uniform decimation with maximum absolute error at offset two. Darker areas indicate higher error values.

for producing effective representations. Because the subdomains we use are directly tied to the error regions, we refer to them by the *offset* between a low resolution and its error resolution.

4.1.4 Error Tolerance

A function is required to determine whether a subdomain should be assigned to a low resolution or a higher one. In practice, this is a simple comparison between the error for the region and the user-specified error tolerance. This could be replaced with another predicate function for more complicated comparisons. For example, a more advanced error tolerance could take both average and maximum absolute error into account.

4.2 AR Tree Construction

Octrees for adaptive resolution representations can be constructed with or without a target error tolerance. Both methods use the same tree structure and can potentially be traversed dynamically to build a new representation with a different tolerance.

General AR Tree Construction

Each node of the AR Tree represents an area or volume in the domain, with the root node representing the entire domain. Each node contains a reference to a subvolume at one level in the multiresolution hierarchy (the data), and the error value associated with it. Each child node represents a subvolume of its parent's volume.

Starting with the full volume of the data set at the lowest resolution, the volume is subdivided along the boundaries of the error set regions (eight children in the case of an octree; four if it's a quad tree). Each node is assigned its volume in the lowest resolution with the error value associated with it. This is repeated until the volume cannot be divided further at this resolution (there is only a single error value for the region). The volume is subdivided and the child nodes are moved to the next higher resolution. This process is

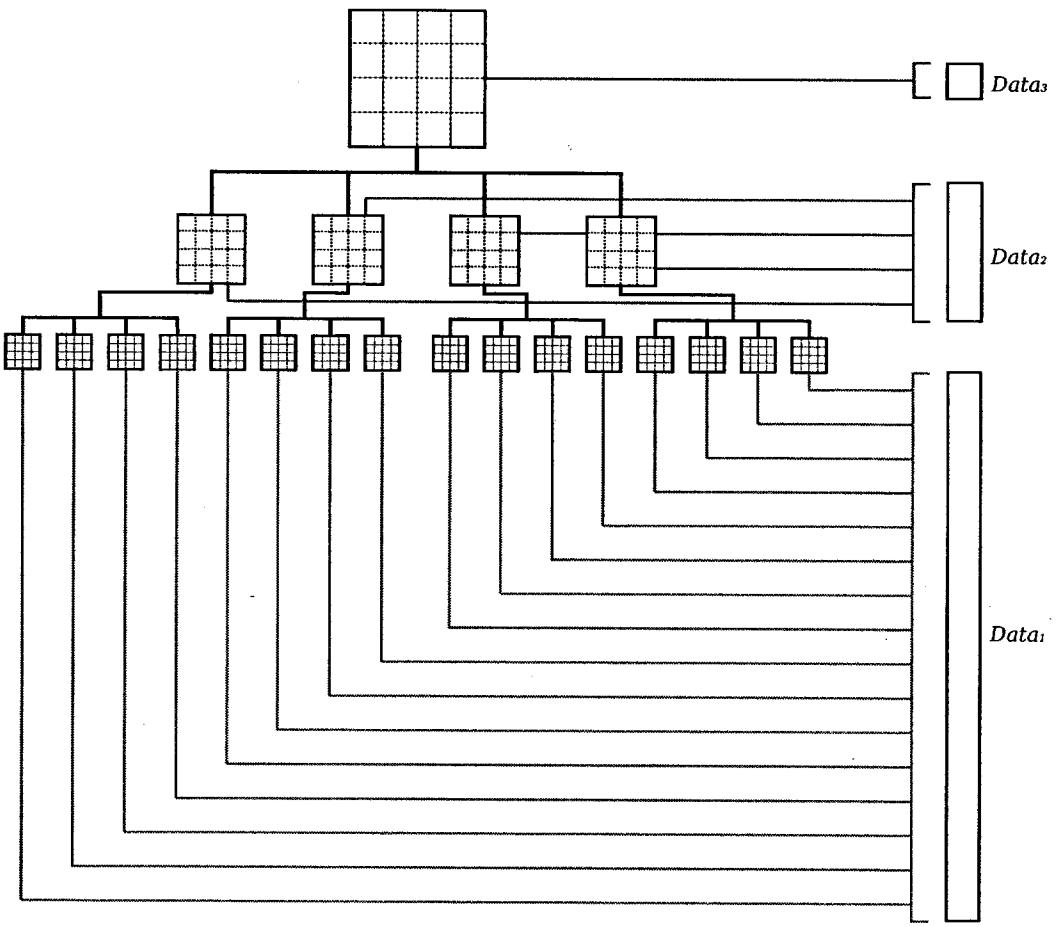


Figure 4-2: An octree representing the domain of the data set. Every level of the tree stores an entire representation of the domain at some resolution. Child nodes represent a subvolume of its parent. Each node contains a reference to the data for the subvolume.

repeated for each subvolume until the data for the subvolume is at the original resolution and no further subdivision is necessary.

Trees constructed in this manner are full trees, in that every leaf of the tree is at the original resolution. By traversing (pruning) this tree with an error tolerance, adaptive resolutions can be produced. This would be useful in an interactive scenario where a scientist could "play" with the error tolerance, while zooming into regions in the data set. An adaptive resolution representation generated with an extremely low error tolerance could possibly result in the original data set. Conversely, a very high error tolerance could result in the lowest uniform resolution data set.

4.2.1 AR Tree Construction with an Error Tolerance

Similarly, AR trees can be constructed with a target error tolerance. This process is the same as the general AR tree construction except that each node in the tree is tested against the error tolerance before expansion. If the data in the volume is within the error tolerance, the node is not expanded. The result is equivalent to traversing a tree constructed without an error tolerance and pruning the children of nodes that meet the error tolerance. This method could be useful in situations where the user has some desired floor tolerance. It is still possible to traverse such a tree to produce adaptive resolutions with greater error tolerances.

4.3 AR Tree Traversal

Once an AR tree has been constructed, the tree can be traversed based on a given error tolerance. In the case of a tree constructed with an error tolerance, it is only possible to build additional adaptive resolutions with larger error tolerances.

For each node in the tree, the error value associated with its data block is tested against the tolerance. If the data block meets the tolerance requirements, that block is incorporated into the result. Otherwise, the test is applied to the children for the node. If the node has

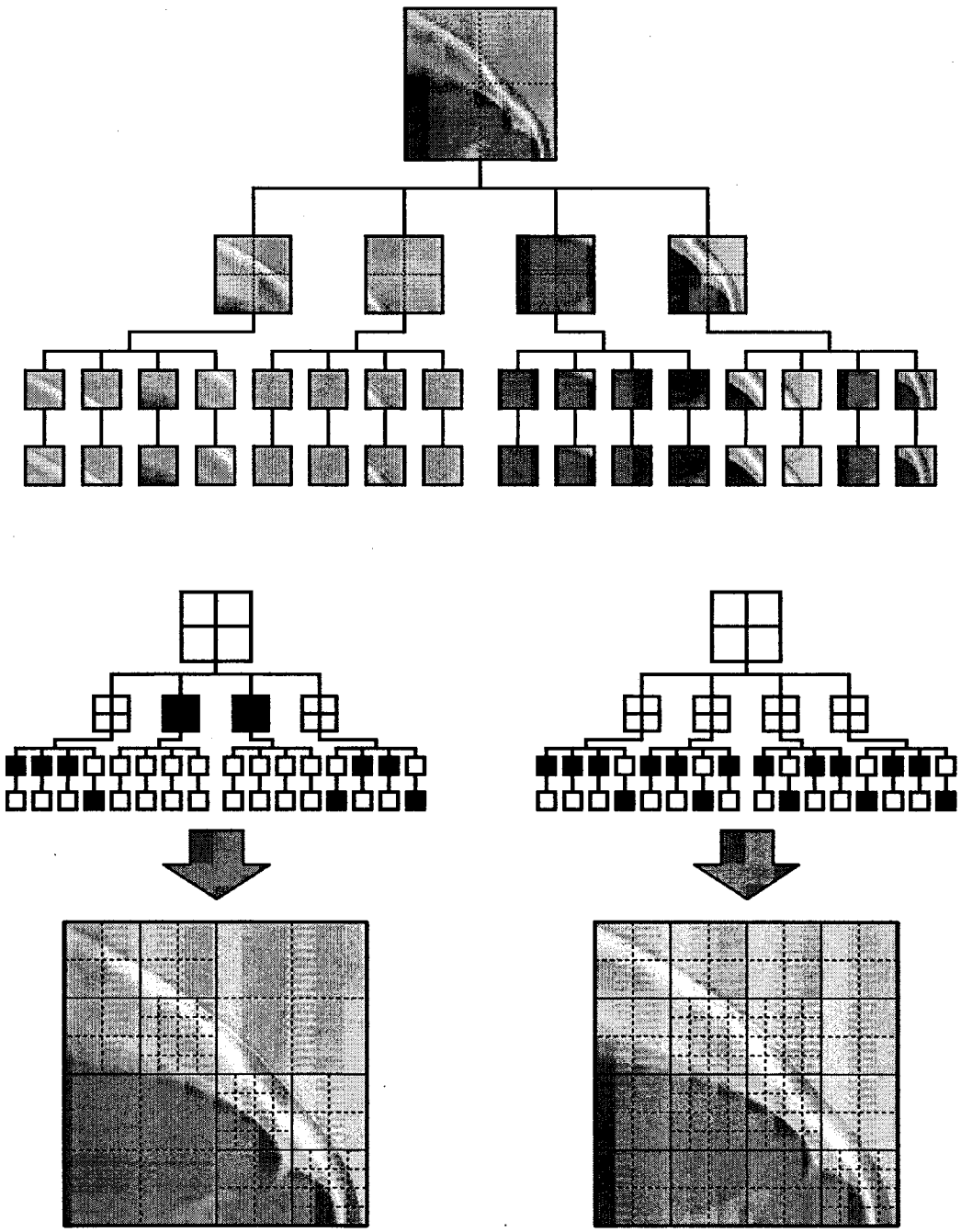


Figure 4-3: An adaptive resolution tree and two possible representations generated from it with two different error tolerances.

no children, the subvolume is at the highest resolution possible. In the case of a full AR tree, this occurs when the node is at the original resolution.

4.3.1 Tree Simplification

Once the tree has been pruned to the error tolerance, an optional pass can be made to simplify the structure of the AR tree. For example, if all of the children of a node are placed at the same resolution, the volumes of the children can be combined into a single block for the parent node (Figure 4-4). These methods can also involve more complicated, heuristic approaches, such as raising the resolution of a few small, low resolution volumes to match that of its siblings (Figure 4-5).

Reducing the size of the tree should also reduce the overall cost of the adaptive resolution. If the tree is used in an interactive setting, reducing the number of nodes also reduces the number of issued read commands. When the adaptive resolution is generated offline, simplifying the tree reduces the number of separate meshes needed when the data set is rendered.

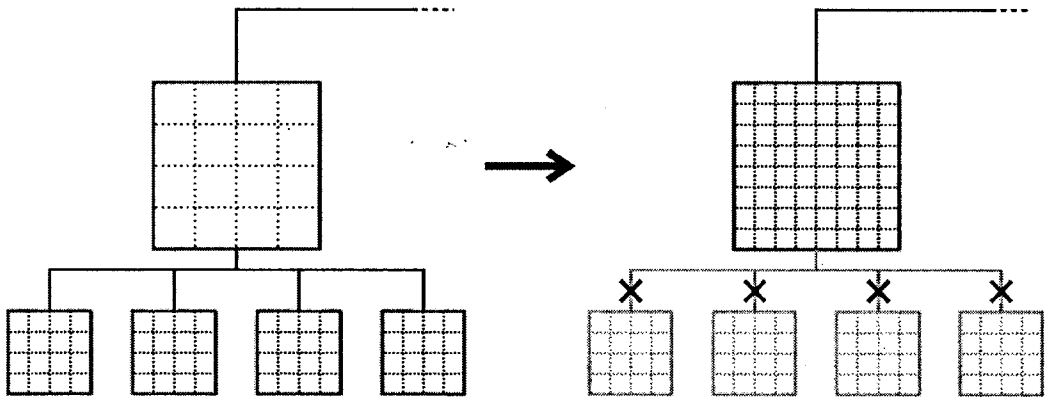


Figure 4-4: The child nodes can be collapsed into the parent if they are all leaves at the same resolution level.

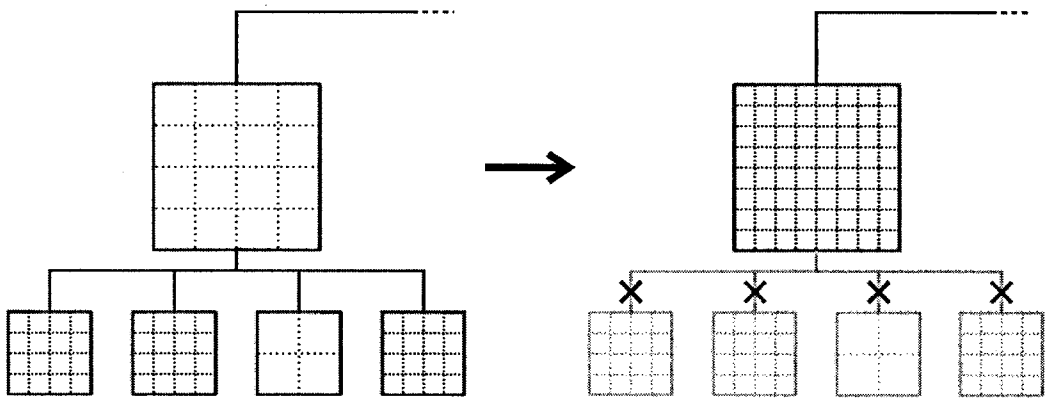


Figure 4-5: One child node's resolution is raised to match its siblings. The child nodes can then be collapsed into the parent to simplify data access.

CHAPTER 5

RESULTS

With the techniques described in Chapters 3 and 4 multiresolution and adaptive resolution representations can be constructed from a large, uniform data set. To understand what benefits these offer over the original data, we evaluated the performance of these new representations compared to the original.

Unlike a uniform resolution, multiresolutions and adaptive resolutions use several parameters that can affect their performance. In multiresolution data sets, the user specifies the number of levels in the hierarchy and the resolutions for the data and error sets. For adaptive resolutions, these factors include:

- The decomposition method used to generate the low resolutions from the original data set.
- The size of the subdomains used to calculate error and assign resolution levels.
- The error function and tolerance value used to generate the adaptive resolution.
- The tree simplification parameters used to reduce the number of meshes in the final representation.

Our evaluation focuses on how subdomain sizes and error tolerance affect the performance of adaptive resolution. The two decomposition methods we implemented (uniform decimation and wavelet decomposition) produce low resolutions of nearly the same size. Therefore, the use of one over the other is not expected to affect the performance significantly. Tree simplification methods were not compared because their effect can vary

depending on the data set and the impact of the number of meshes can be seen by varying the resolution offset. In all our adaptive resolutions, nodes whose siblings are all at the same resolution are collapsed into their parents, as described previously.

5.1 Evaluation Methods

5.1.1 Data Sets

The data set used in our evaluation consists of 30 time steps of simulation data from the OPEN GEOSPACE GENERAL CIRCULATION MODEL (Open GGCM) [18]. Each time step contains a volume of $630 \times 200 \times 300$ floating point values. One time step is roughly 145 MB in size, making the total size of the data set around 4.2 GB. Since each time step can easily fit into memory, this is not the best example for demonstrating the benefits an adaptive resolution offers over the original. With this in mind, a second data set was generated by scaling up the original to $630 \times 400 \times 600$. At 577 MB, each time step can still fit into main memory, but less space is left for calculations made during visualization. The total size of this data set is 16.9 GB.

Figure 5-1 shows two pseudocolor slices from the first and last time steps of the data set. From these we can see that there are large portions of the data set that could be represented at lower resolutions. This is already done to some degree through the perimeter lattice used in simulation. This grid increases sampling in areas where interesting behavior is expected, leaving areas such as the edges and the left-hand side of the image at lower resolutions. The grid is fixed through all time steps, while our adaptive resolutions are generated separately for each one. Because of this, we can expect from comparing these two images that earlier time steps will be more easily reduced in adaptive resolutions than later ones.

5.1.2 Parameters

The adaptive resolutions we used were constructed from a multiresolution data set of four levels (including the original) generated through uniform decimation with error sets calcu-

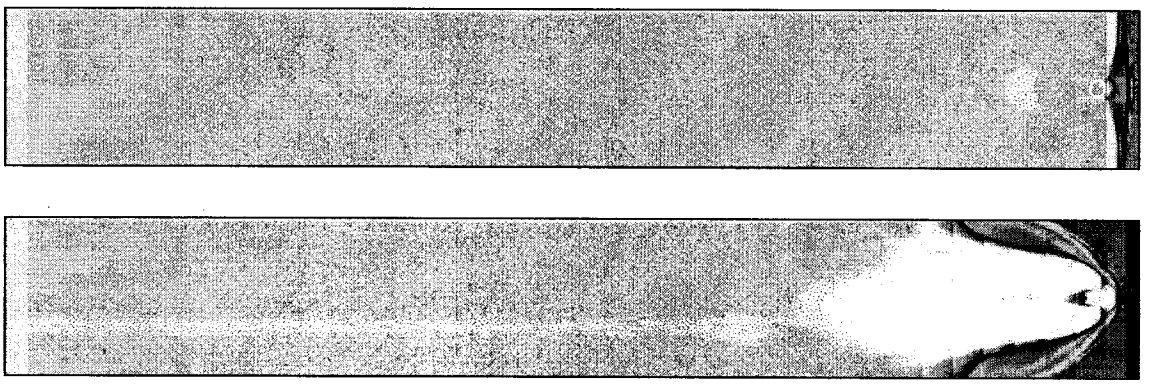


Figure 5-1: Two pseudocolor slices from the first and last time steps from the simulation data.

Resolution	Small Data Set	Large Data Set
0	$630 \times 200 \times 300$	$630 \times 400 \times 600$
1	$315 \times 100 \times 150$	$315 \times 200 \times 300$
2	$158 \times 50 \times 75$	$158 \times 100 \times 150$
3	$79 \times 25 \times 38$	$79 \times 50 \times 75$
4	$40 \times 13 \times 19$	$40 \times 25 \times 38$
5	$20 \times 7 \times 10$	$20 \times 13 \times 19$
6	$10 \times 4 \times 5$	$10 \times 7 \times 10$
7	$5 \times 2 \times 3$	$5 \times 4 \times 5$
8	$3 \times 1 \times 2$	$3 \times 2 \times 3$
9	$2 \times 1 \times 1$	$2 \times 1 \times 2$
10	$1 \times 1 \times 1$	$1 \times 1 \times 1$

Table 5.1: Low resolution sizes for each test data set. Resolutions lower than 3 are only used for error data.

Res. Offset	Small Data Set	Large Data Set
2	75,050	296,250
3	9,880	38,000
4	1,440	4,940
5	200	700
6	30	100
7	6	18

Table 5.2: Maximum number of meshes (subdomains) possible for each resolution offset.

lated using a maximum-absolute error function. The choice of error function is important as it defines the subvolumes of the domain that are most important to the user. Maximum-absolute error is good for capturing the largest error values in the data set but has the disadvantage that small spikes can overpower the error for an entire region.

From the smaller data set, we constructed 20 adaptive resolution representations of our data set using four resolution offsets and five error tolerances. We used resolution offsets 2, 3, 4, and 5 to build the subdomains. The error tolerances (0.5, 1, 3, 7, 15) roughly correspond to 0.5, 1, 5, 10, and 20 percent error. For the larger data set, we constructed 25 adaptive resolutions, using offsets 3, 4, 5, 6, and 7 and the same error tolerances.

5.1.3 Timings

All timings were made on a commodity desktop with an Intel Core 2 Quad processor (4×2.40GHz) and 2GB of RAM. We used VisIt [2] to render an image for each time step of the original and adaptive resolution data sets. The total engine time and time to render one frame were noted individually for each time step. Engine time includes the time for the I/O operations to read the data set into memory, the database operations to filter the data, and the overhead from its socket interface. Although the desktop had multiple cores, we used the serial version of the VisIt engine for simplicity.

This was done for three plot types; a pseudocolor slice along the Y-axis, a diagonal pseudocolor slice, and a volume rendering by splatting (Figure 5-3) of the full domain.

In order to use VisIt, our uniform and adaptive resolution representations are written

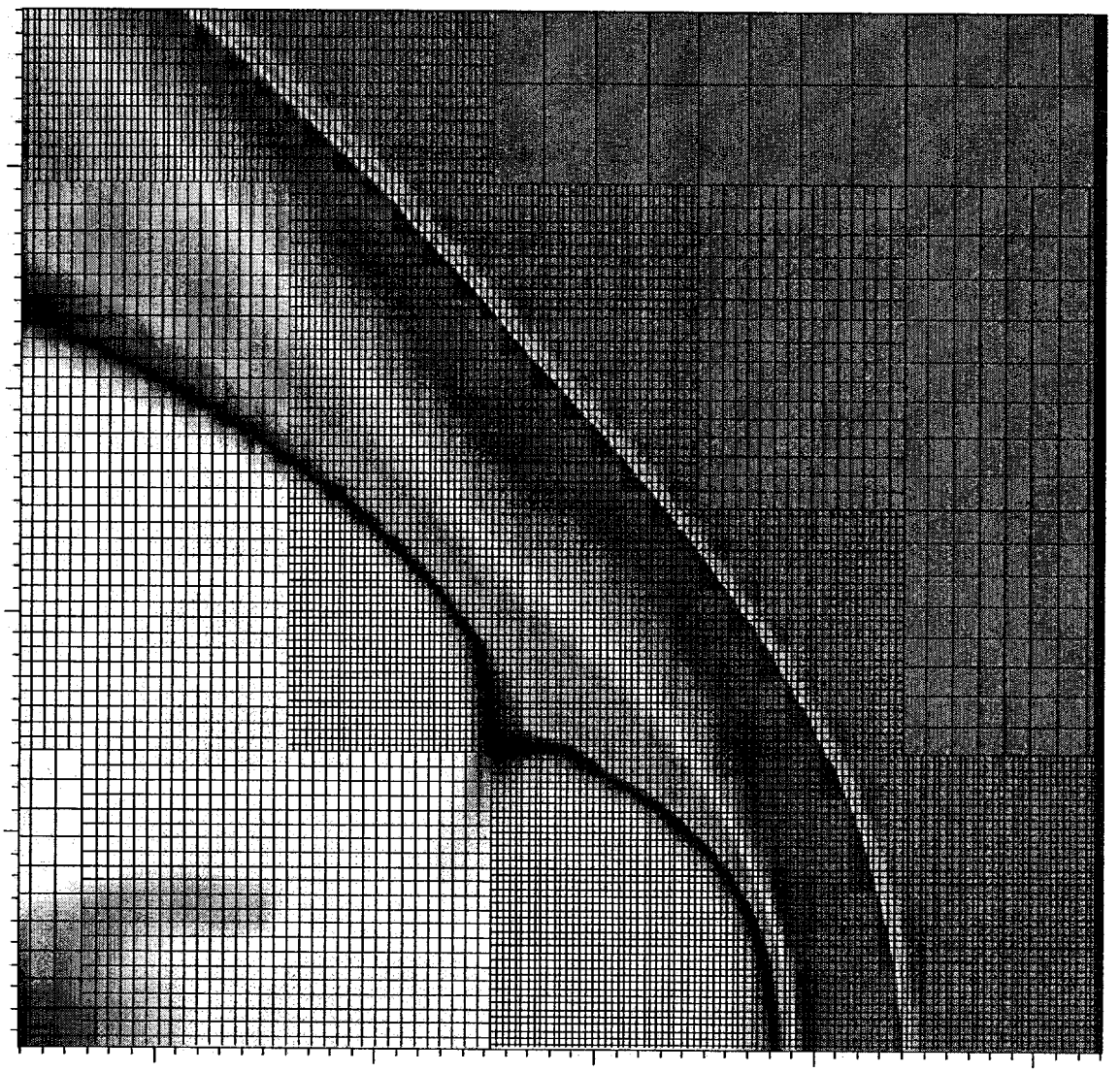


Figure 5-2: Detail of a pseudocolor slice plot of an adaptive resolution representation with a grid overlay. Four resolution levels are present in this image.

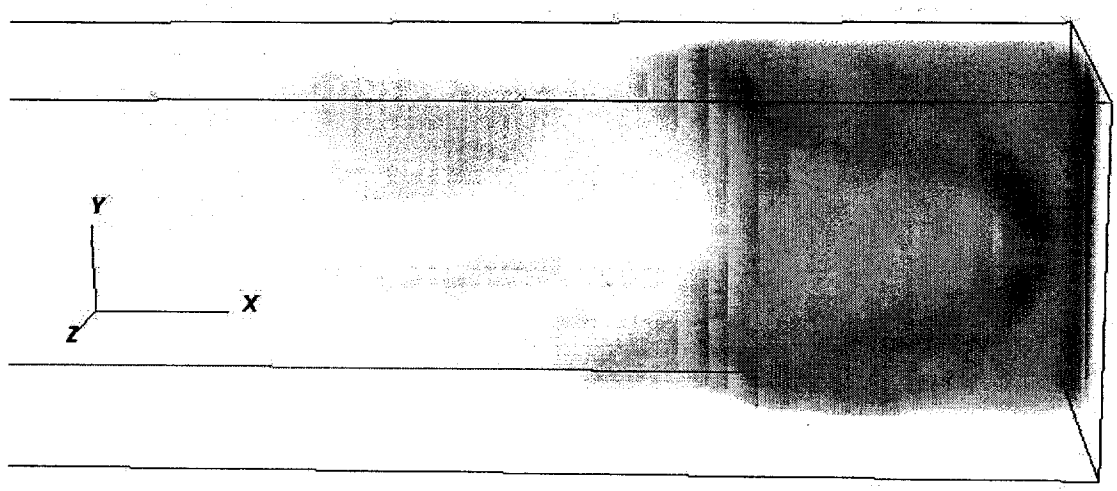


Figure 5-3: A partial volume rendered (splatting) plot of an adaptive resolution representation.

out in Silo format [1]. Each subdomain is added as a single mesh and variable to a multi-mesh and multi-variable structure. For uniform, multiresolution data, only one Silo mesh and variable for the entire domain were added.

Finally, we have very little control over how VisIt and Silo choose to read and render our representations. For the purposes of these evaluations, VisIt is treated as a "black box" where the only element we can directly control is the data we give it.

5.2 Multiresolution Demonstration

Multiresolution data sets are more than just an intermediate step between a large, uniform resolution and an adaptive representation. They can be used on their own for visualization and exploration of the domain. Figure 5-4 shows three slices taken from low resolution data sets along with the corresponding slices from the error sets. For comparison, each error set is at the same resolution and rendered with the same color scale, meaning that each was created using a different resolution offset from the original data. The result is three error

sets of the same size that represent the error for three data resolutions. As expected, we can see from the darker areas in the images the error becomes greater with lower resolutions.

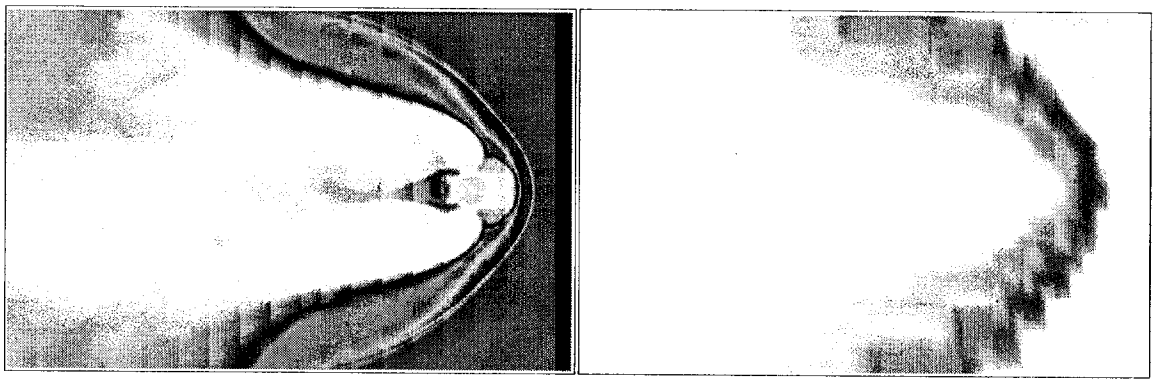
Figure 5-5 shows a box plot of combined timings for rendering pseudocolor slices from two data sets (data and error) at varying resolutions. From the plot, we can see that there is very little variance between the time steps for each data/error pairing. Also, there is little difference among the error offset choices for each low resolution data set. Finally, the changes in overall time after resolution 1 are not very dramatic. From these times, we see that using low resolution data with error information is a reliable way of decreasing the engine and render times. This demonstration also shows us that low resolutions with error information have value on their own for getting broad overviews of the domain and examining some areas in more detail.

5.3 Adaptive Resolution Evaluation

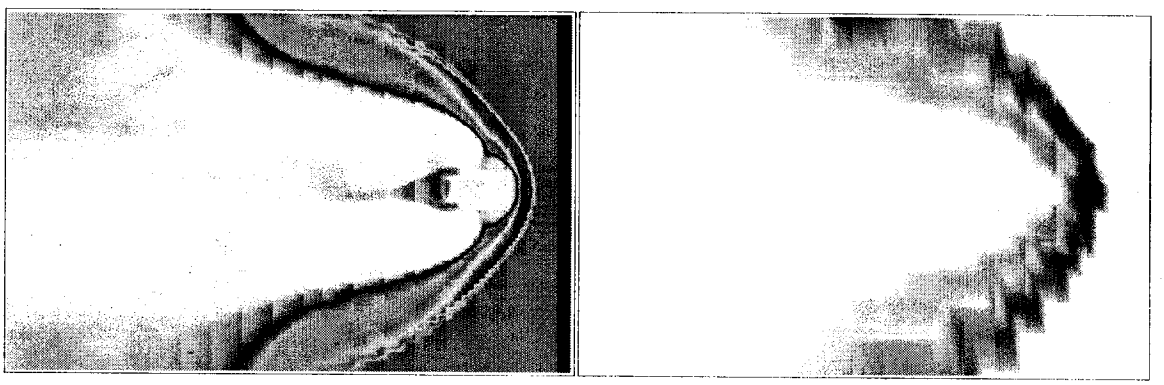
5.3.1 630x200x300 Data Set

Figure 5-6 is a box plot of the combined engine and render times for volume rendered plots of all individual time steps, grouped by the parameters used to generate the adaptive resolutions. The timings for the original uniform resolution are also included for comparison. For each grouping, the box indicates the middle 50 percent around the median (line), while the whiskers show the upper and lower quartiles. The shaded box represents the 95 percent confidence interval for the mean. Outliers are shown with a circle. From these plots we can see that the performance of an adaptive resolution depends greatly on the parameters used to generate it. As expected, higher error tolerances are associated with lower total times and many subdomains incur significant overhead in the representation. We also begin to see the predicted slow down from large subdomains between resolution offsets 4 and 5.

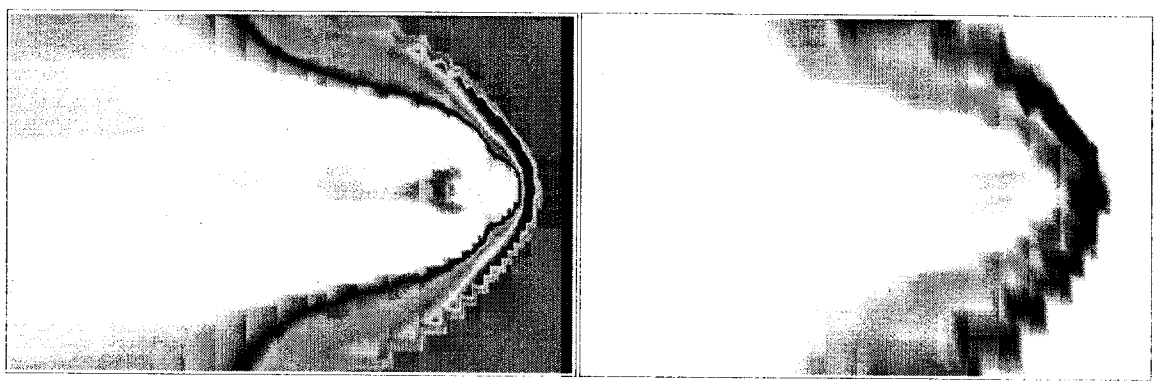
Figures 5-7 and 5-8 show the separate engine and render times for volume rendered plots. Although the render times appear to be affected by the parameters of an adaptive resolution, engine time accounts for most of the overall time. Figure 5-9 demonstrates this



D_1 $E_{1,4}$



D_2 $E_{2,4}$



D_3 $E_{3,4}$

Figure 5-4: Slice renders of three low resolution data sets generated through uniform decimation with maximum absolute error. Darker areas indicate higher error values.

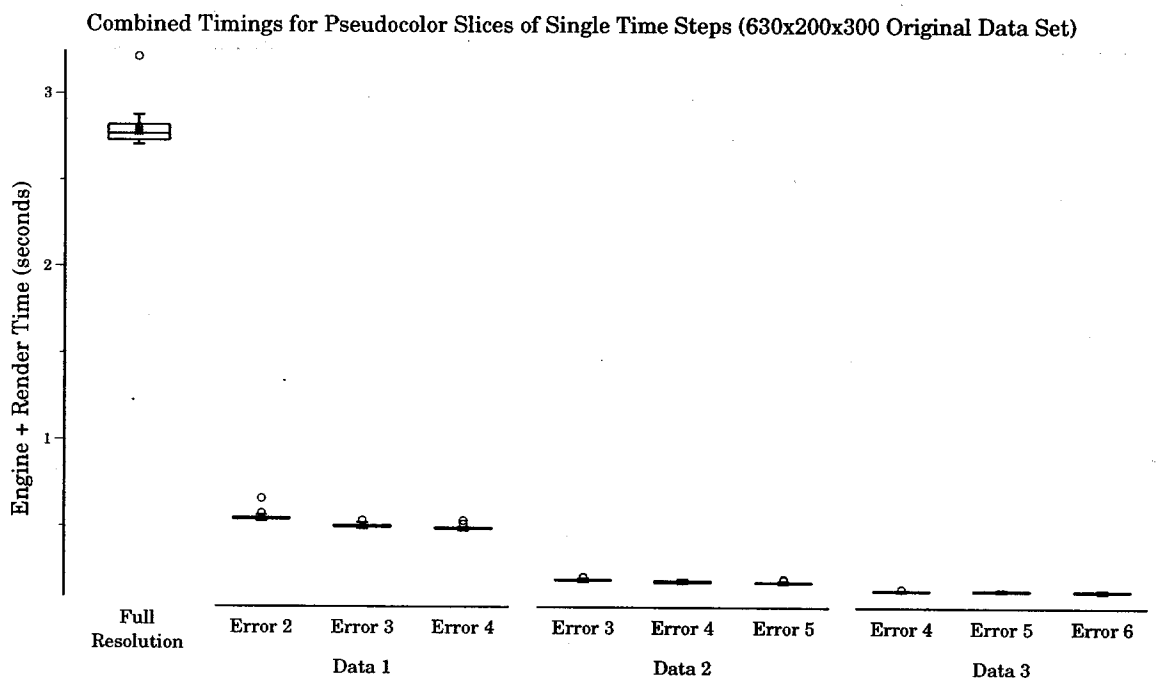


Figure 5-5: Total times for pseudocolor slice plots of a data set with error rendered along side.

by showing the render and engine times by time step for adaptive resolutions constructed with resolution offset 4 and error tolerances 1.0, 3.0 and 7.0. We can also see from the error bars indicating the 95 percent confidence interval for the mean times that engine time can vary greatly while render times remain relatively constant.

The most striking difference between adaptive resolution and the original resolution is the wide range of observed times for adaptive resolution and the extremely narrow range for uniform resolution. This can be attributed to some time steps being more easily reduced in adaptive resolution than others, as discussed previously. Figure 5-10 shows the combined times for each time step. Again, earlier time steps in the simulation are simpler and can be reduced to lower resolution with less error. As the simulation continues, interesting behavior fills more of the domain, forcing more subdomains to be kept at higher resolutions. Figure 5-11 also shows this trend.

Finally, we also recorded rendering times for each time step using two pseudocolor slices:

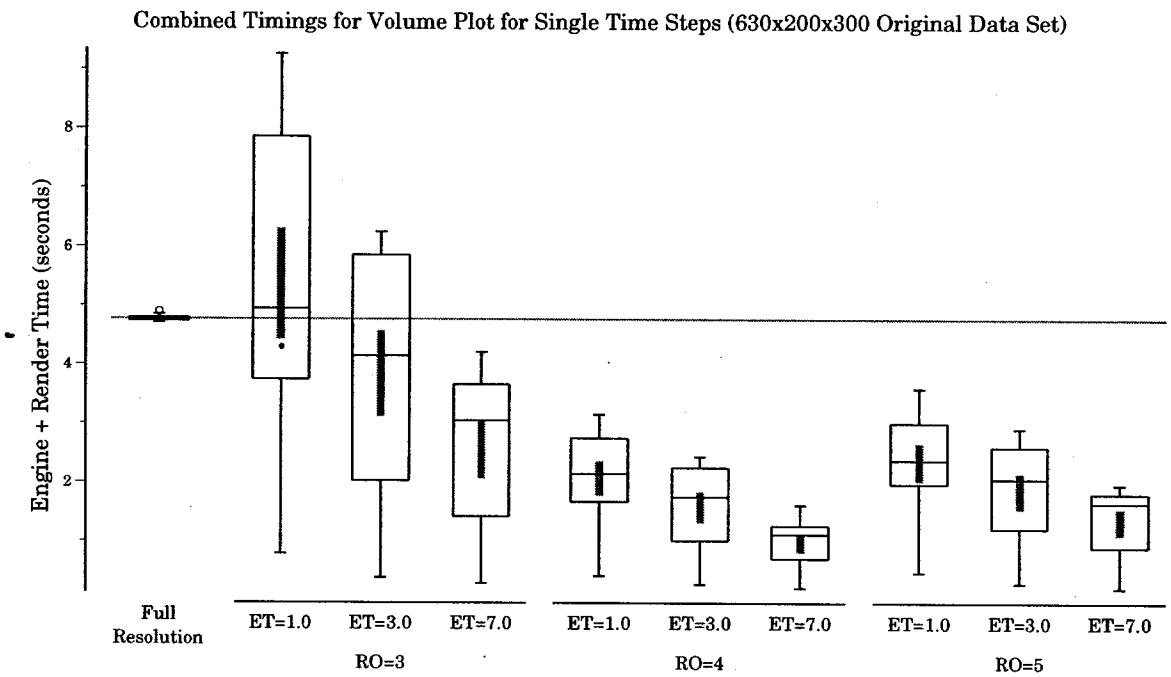


Figure 5-6: Total times for volume render plots of individual time steps at adaptive resolutions with different resolution offsets (RO) and error tolerances (ET) with the original data set for comparison (Maximum Absolute Error).

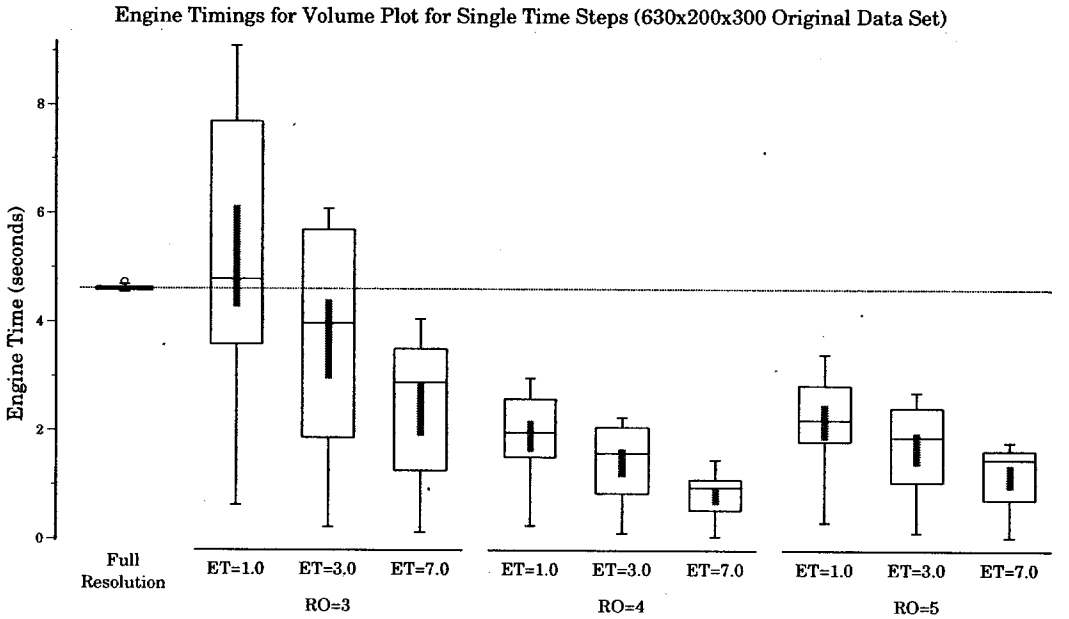


Figure 5-7: Engine times for volume render plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

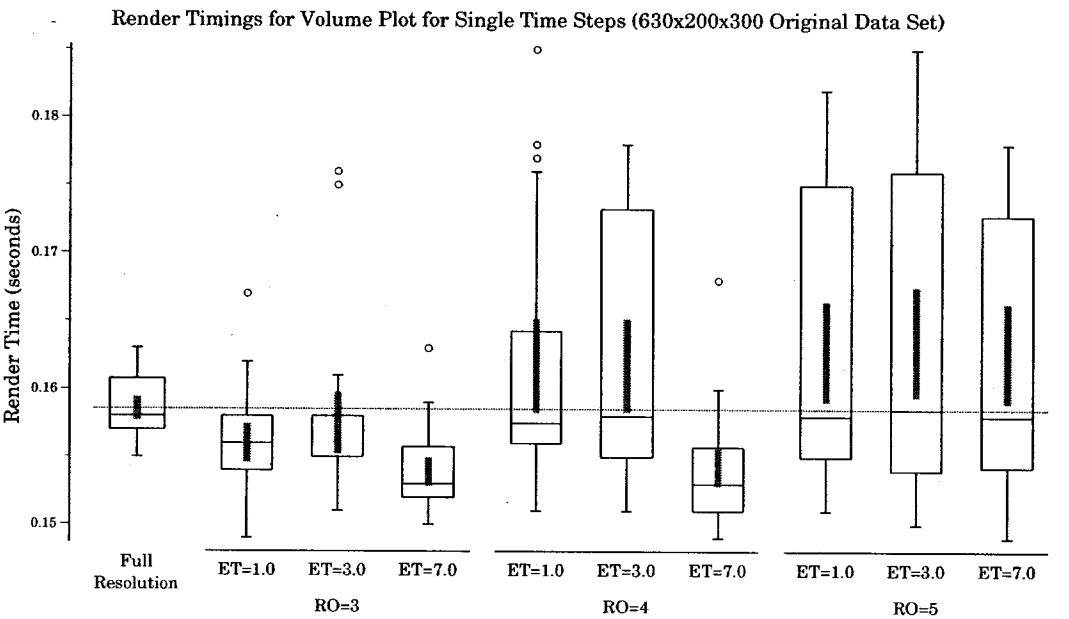


Figure 5-8: Render times for volume render plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

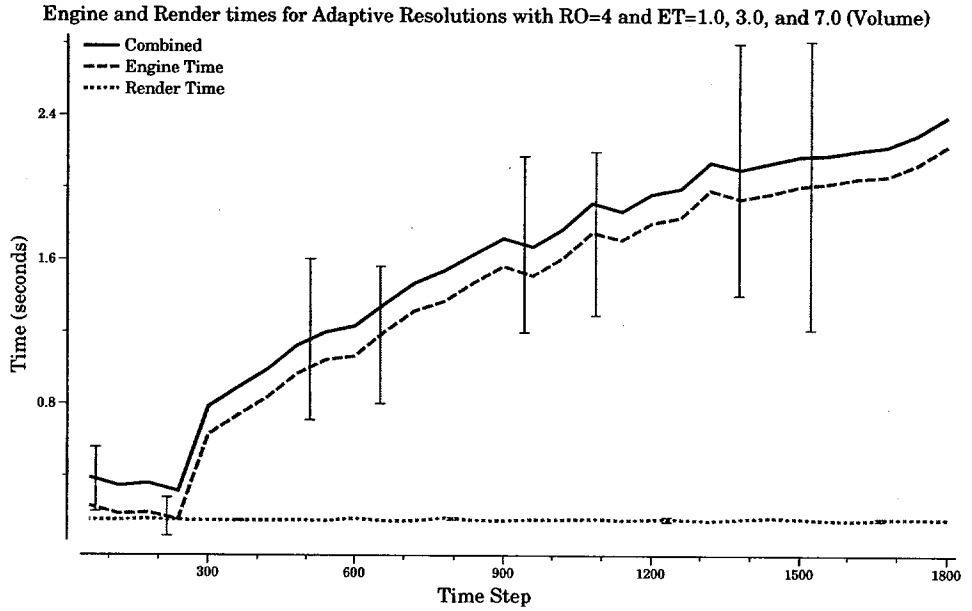


Figure 5-9: Engine and render times by time step for adaptive resolutions with a resolution offset of 4 (Maximum Absolute Error).

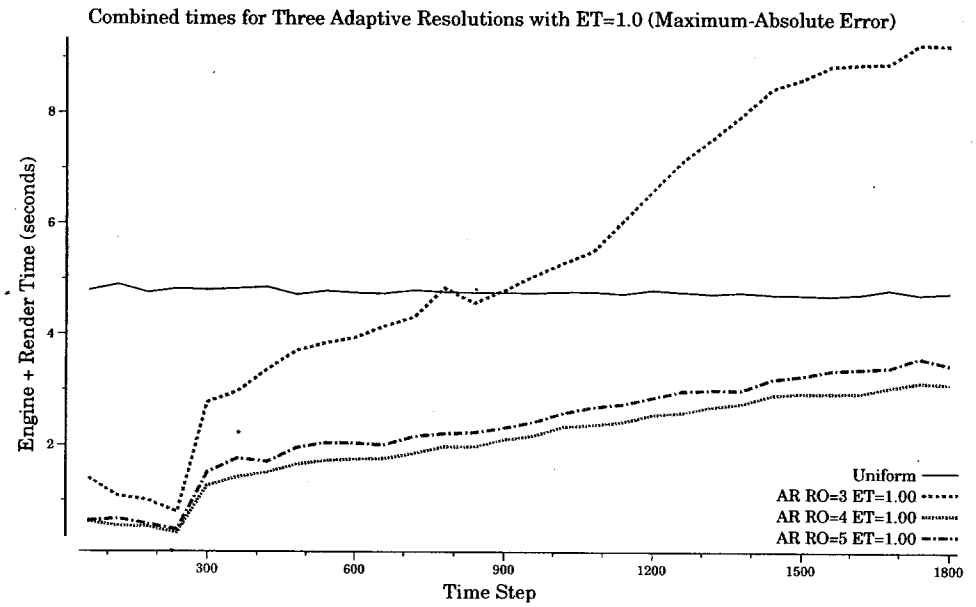


Figure 5-10: Total time for each time step for adaptive resolutions generated with the same error tolerance (1.0) and varying resolution offsets (Maximum Absolute Error).

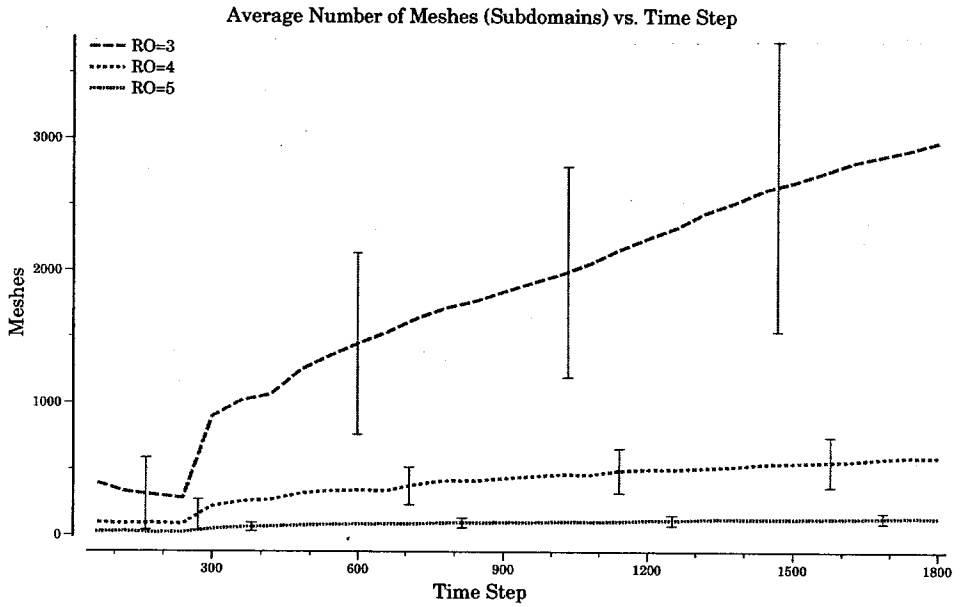


Figure 5-11: Average number of meshes (subdomains) for each time step organized by resolution offset. Each grouping includes all 5 error tolerances (Maximum Absolute Error).

across the Y-axis (5-12), and diagonally (5-13). In these two cases, engine time represents not only the I/O involved in reading the data set into memory but the filtering process for the slice operator. Uniform resolution has an advantage for the Y-axis slice, since the pertinent data can be retrieved more easily compared to the multi-mesh adaptive resolution. Figure 5-14 shows the mean engine and render times for a slice plot from the same adaptive resolutions used in Figure 5-9. Again, engine time is the main contributor to the overall time, but render time is greater and varies more than before. When the slice is taken diagonally, the results are similar to the volume rendering timings. Despite the differences, the same patterns of error tolerance and subdomain size affecting the overall performance can be observed.

5.3.2 630x400x600 Data Set

For the larger data sets, we see most of the same features in the results as before. Figure 5-15 show the combined times for volume rendered plots of the larger data set. Again, we

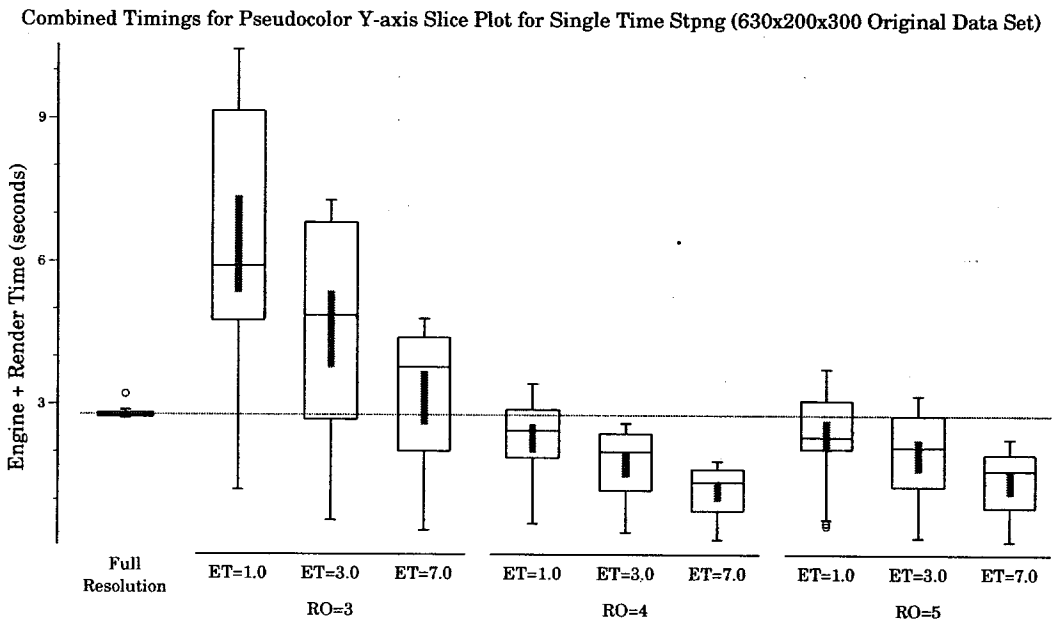


Figure 5-12: Total times for slice (Y-Axis) plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

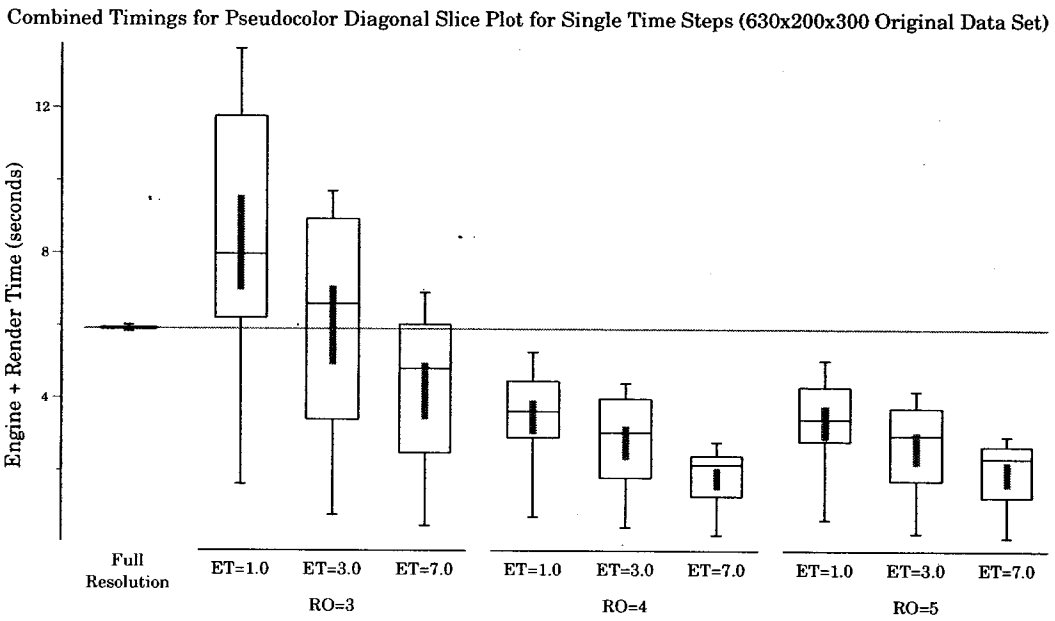


Figure 5-13: Total times for slice (diagonal) plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

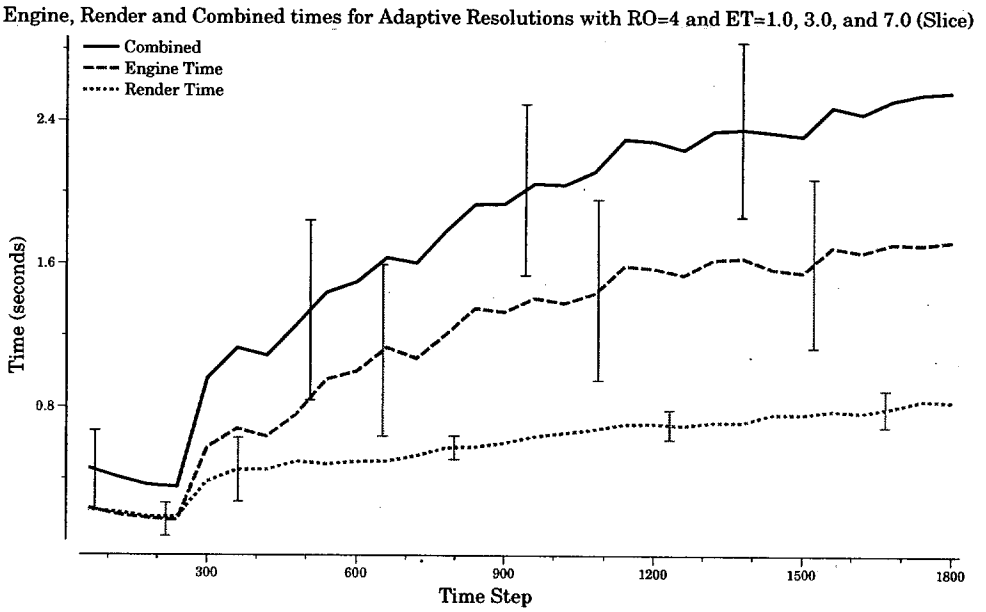


Figure 5-14: Engine and render times for slice (Y-Axis plots) by time step for adaptive resolutions with a resolution offset of 4 and error tolerances 1.0, 3.0 and 7.0 (Maximum Absolute Error).

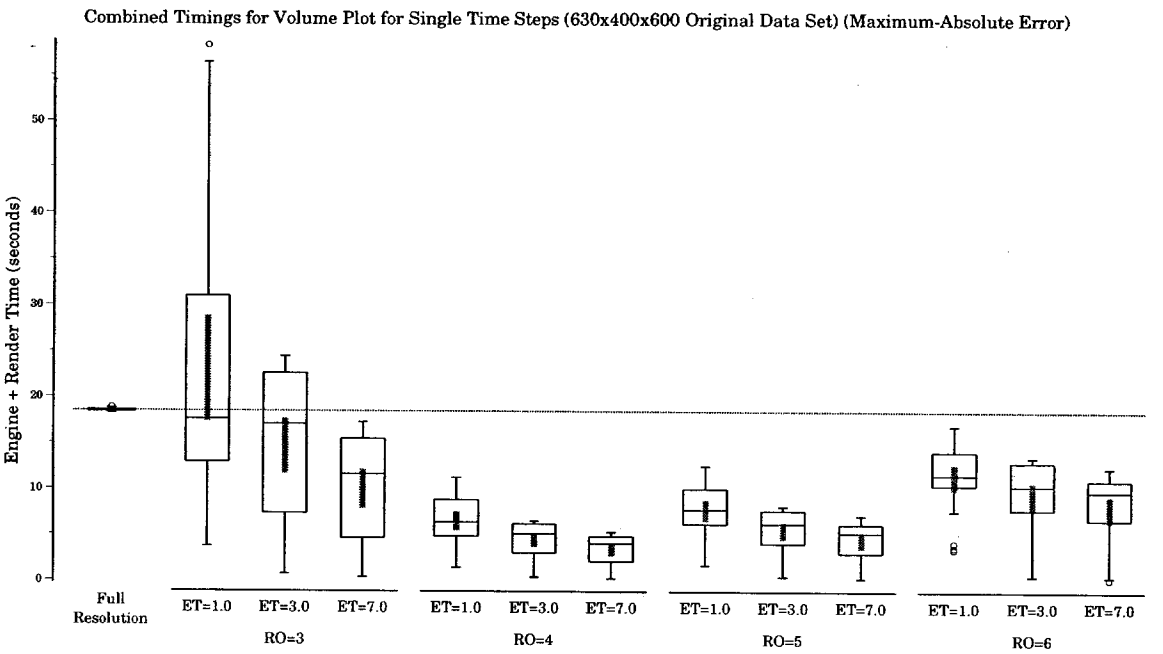


Figure 5-15: Total times for volume render plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

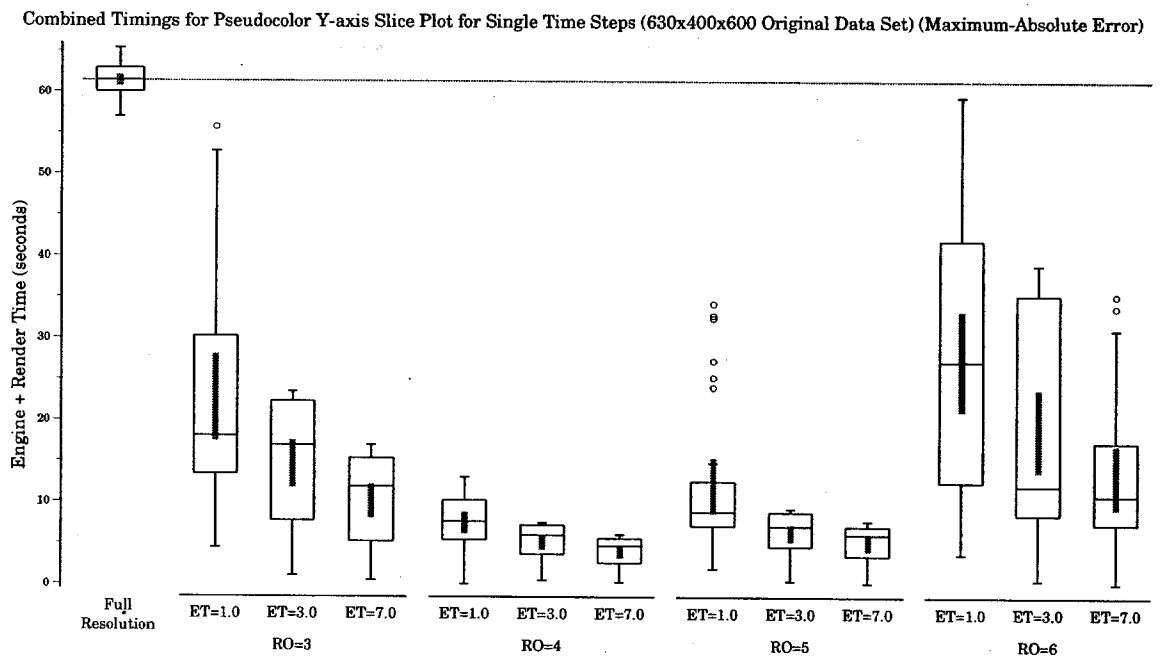


Figure 5-16: Total times for slice (Y-Axis) plots of individual time steps at adaptive resolutions (Maximum Absolute Error).

see that times for adaptive resolutions vary greatly compared to the uniform resolution and adaptive resolutions with lower error tolerances have lower times. The slow down from large subdomains can be seen more clearly with the addition of resolution offset 6 and the cost of many, small subdomains is still present with resolution offset 3.

In the slice rendering test case (Figure 5-16), we see a drastic rise in the combined times from the smaller data set (Figure 5-12) for the original, uniform data set. This is caused by the paging that occurs during the test, despite the fact that the larger time steps are still small enough to fit in main memory. Adaptive representations follow the same patterns as before.

5.4 Average Absolute Error

Maximum absolute error captures the largest differences between the original and reconstructed data sets. Because of this, the values of large error regions can be dominated by a few high error values. Average absolute error can be used to reduce the impact of these high values in the error set and in the adaptive resolution.

Figures 5-17 and 5-18 show combined timings for slice and volume plots rendered from adaptive resolutions generated with average absolute error. Compared with the adaptive resolutions generated with maximum absolute error, we see faster times for lower error tolerances. In fact, some of these adaptive data sets reach our goal of interactive time (< 1 Second per time step) for the whole time series.

Another important property of this error function is that error values tend to decrease as the size of the error region increases and move values are included in the error calculation. The effects of this can be seen in resolution offset 5 where the times do not begin to increase as they had with maximum absolute error. Interestingly, the adaptive resolutions generated with error tolerance 3.0 using offsets 4 and 5 are actually entirely at the lowest resolution possible.

5.5 Summary

In these evaluations we have seen both multiresolution and adaptive resolution data sets achieve faster times than the full resolution data set. For adaptive resolutions, we have learned that the parameters used to build the representations have a strong effect on their performance.

Although many individual time steps can be said to have reached interactive time, only a few the results from our adaptive resolution data sets are entirely within this range. However, there are many aspects of our evaluation set up that could be improved. First, the error tolerances we used were stringent. Using more relaxed error tolerances should improve the performance. Also, all of our engine times were made using the serial version

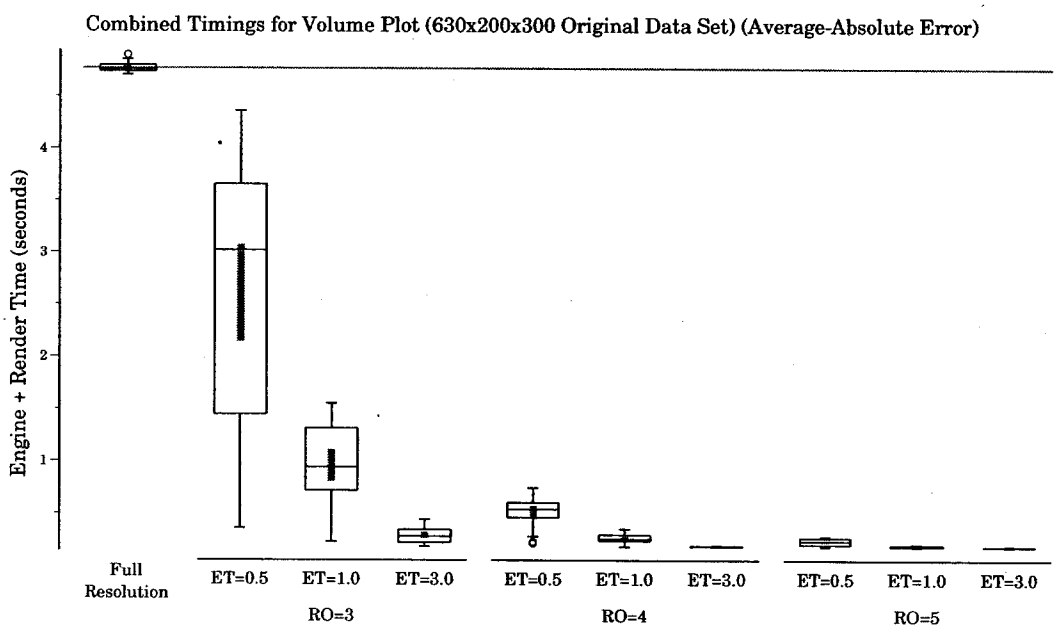


Figure 5-17: Total times for volume plots of individual time steps at adaptive resolutions generated using average absolute error.

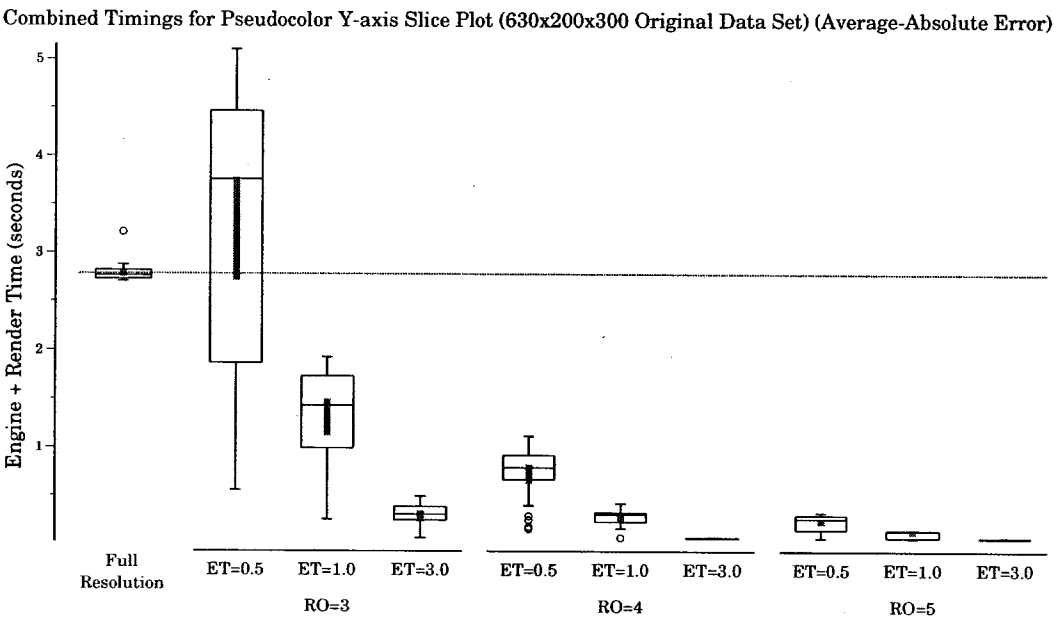


Figure 5-18: Total times for slice (Y-axis) plots of individual time steps at adaptive resolutions generated using average absolute error.

of the VisIt engine. Adaptive resolutions should benefit greatly from parallel environments as each subdomain can be processed individually, in parallel. Finally, these evaluations were run using commodity software on a modest desktop. Improved times would certainly be achieved with a specialized rendering tool running on a more powerful workstation.

CHAPTER 6

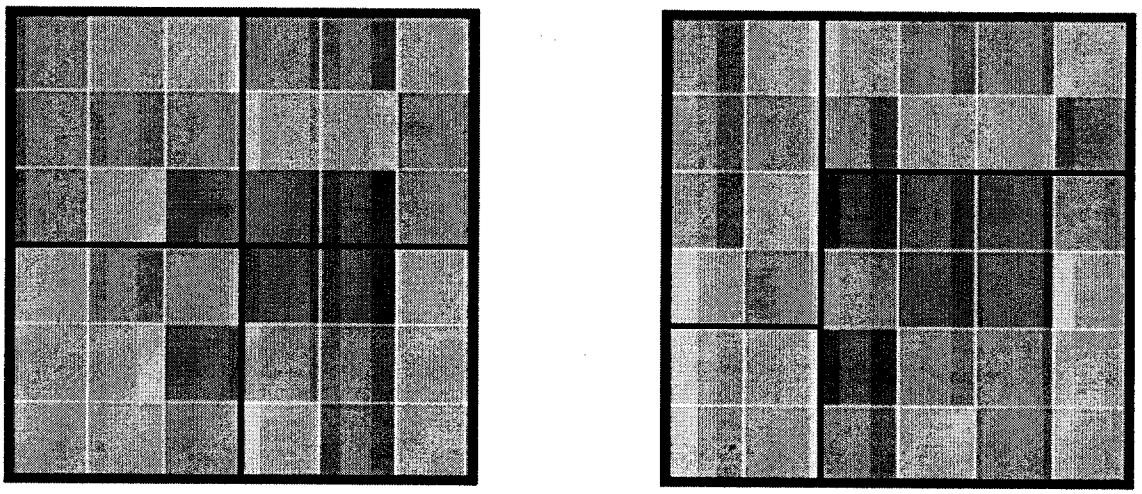
CONCLUSIONS

6.1 Conclusions

The goals of this thesis were to develop an error model for calculating and representing local error from data reduction, use this model to develop tools for generating multiresolution and adaptive resolution data sets, and evaluate the performance of these representations. This paper has described a flexible error model that can be used to generate low resolution data sets with error information. We have presented a process for building adaptive resolutions from this data and evaluated their performance. We have seen from the results that adaptive resolutions generated with the proper parameters can achieve faster overall times compared with the original uniform resolutions, while still providing a useful rendering of the data. In addition, error information generated simultaneously with the low resolution can also be visualized to help the user understand the authenticity of the low resolution data.

6.2 Future Work

The process of constructing adaptive resolutions as described in this paper can be improved in several ways. We have not explored in great detail the many possible tree simplification methods and their effects on performance. Another way to improve the use of subdomains is to create them more intelligently. Currently, the domain is subdivided without any regard to the contents of the underlying error set. A kd-tree could replace the existing octree and be used to partition each subdomain according to error values. This may improve



(a) (b)

Figure 6-1: Two domains partitioned into four subdomains. Colored squares represent error regions with darker colors indicating higher error. (a) Split around a central point. (b) Split along planes with regard to error values.

performance by avoiding cases where regions of high error are split across several nodes in the tree (Figure 6-1).

In the future, the ideas presented in the paper could be incorporated into an interactive system either built as a stand-alone tool or plug-in for an existing visualization environment. The user would be given controls to choose the error tolerance and other parameters for a given data set. The representations could be generated offline and stored until needed.

Ultimately, these adaptive resolutions should be generated dynamically for the user. An adaptive resolution tree can be kept by the visualization tool. When the user changes a parameter, the required data is fetched from the multiresolution hierarchy and the adaptive resolution is rebuilt. Through this interaction users can find the best parameters for exploring their data set.

BIBLIOGRAPHY

[1] Silo: a mesh a field I/O library and scientific database. Lawrence Livermore National Laboratory, Livermore, CA. See http://wci.llnl.gov/codes/silo/index.html.

[2] VisIt visualization environment. Lawrence Livermore National Laboratory, Livermore, CA. See http://www.llnl.gov/visit/home.html.

[3] John J. Benedetto and Michael W. Frazier, editors. *Wavelets, Mathematics and Applications.* CRC Press, Boca Raton, Florida, 1994.

[4] Charles K. Chui. *An Introduction to Wavelets.* Wavelet Analysis and its Applications – Volume 1. Academic Press, 1992.

[5] A. Ciampalini, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.

[6] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, 1997.

[7] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22(1):37–54, 1998.

[8] J.M. Combes and A. Grossman. Wavelets – time frequency methods and phase space. In Ph. Tchamitchian, editor, *Proceedings of the International Conference*, Marseille, 1987. Springer-Verlag.

[9] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, 1986.

[10] Ingrid Daubechies. Wavelet transforms and orthonormal wavelet bases. In Ingrid Daubechies, editor, *Different Perspectives on Wavelets*, volume 47 of *American Mathematical Society Short Course*, pages 1–33. American Mathematical Society, San Antonio, Texas, January 1993.

[11] David Fang, Gunther H. Weber, Hank Childs, Eric S. Brugger, Bernd Hamann, and Kenneth I. Joy. Extracting geometrically continuous isosurfaces from adaptive mesh refinement data. In *Proceedings of 2004 Hawaii International Conference on Computer Sciences*, pages 216–224, Honolulu, HI, January 2004.

[12] Lori A. Freitag and Raymond M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proceedings of SC99: High Performance Networking and Computing*, Portland, OR, 1999. ACM Press and IEEE Computer Society Press.

[13] Jovan Dj. Golic. Periods of interleaved and nonuniformly decimated sequences. *IEEE Transactions on Information Theory*, 44(3):1257–1260, 1998.

[14] Stephane G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[15] Shigeru Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and Applications*, 13(4):50–56, July 1993.

[16] Alex T. Pang, Craig M. Wittenbrink, and Suresh K. Lodha. Approaches to uncertainty visualization. *Visual Computer*, 13(8):370–390, 1997.

[17] J. Raeder. Global Magnetohydrodynamics - A Tutorial. In J. Büchner, C. Dum, and M. Scholer, editors, *Space Plasma Simulation*, volume 615 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 212–246, 2003.

[18] J. Raeder, J. Berçhem, and M. Ashour-Abdalla. The geospace environment grand challenge: Results from a global geospace circulation model. *Journal of Geophysical Research*, 103(14787), 1998.

[19] Philip J. Rhodes. *GRANITE: A Scientific Database Model and Implementation.* PhD thesis, University of New Hampshire, 2004.

[20] Philip J. Rhodes, R. Daniel Bergeron, and Ted M. Sparr. Database support for multisource multiresolution scientific data. In William I. Grosky and Frantisek Plasil, editors, *SOFSEM 2002: Theory and Practice of Informatics, 29th Conference*, volume 2540 of *Lecture Notes in Computer Science*, pages 99–114, Milovy, Czech Republic, 2002. Springer.

[21] Philip J. Rhodes, R. Daniel Bergeron, and Ted M. Sparr. A data model for adaptive multiresolution scientific data. In *Data Visualization: State of the Art 2003*, pages 257–272. Kluwer Academic Publishers, 2003.

[22] Philip J. Rhodes, R. Daniel Bergeron, and Ted M. Sparr. A data model for distributed multiresolution multisource scientific data. In G. Garin, H. Hagen, , and B. Hamann, editors, *Hierarchical and Geometric Methods in Scientific Visualization*, Heidelberg, Germany, 2003. Springer-Verlag.

[23] Don Speray and Steve Kennon. Volume probes: interactive data exploration on arbitrary grids. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization*, pages 5–12, New York, NY, USA, 1990. ACM.

[24] Chaoli Wang and Kwan-Liu Ma. A statistical approach to volume data quality assessment. *Visualization and Computer Graphics, IEEE Transactions on*, 14(3):590–602, May-June 2008.

[25] G. Weber, H. Hagen, B. Hamann, K. Joy, T. Ligocki, K. Ma, and J. Shalf. Visualization of adaptive mesh refinement data. 2001.

Appendix A

BORDER AND GAP PRESERVATION

A.1 Edge Data Loss

When generating a lower resolution data set, by wavelet or decimation, data at the edges of the data set can be lost. In most applications this is not a serious problem. However, there are some situations where preserving the edge data is important. For example, in a rectilinear grid, losing the edge data may cause obvious size reduction when the data set is visualized. This data loss can occur in several ways.

When using decimation, it is possible, at any resolution r, to lose up to $2^r - 1$ rows of data on the edge, depending on the dimension size. At lower resolutions this number becomes larger and the loss more significant. Wavelets also suffer from edge data loss. When a lower resolution is generated using a wavelet, the borders are contracted by $2^r - 1$ points in the base resolution. In addition to this, if a dimension in the original data set is not evenly divisible by 2^r, there is not enough data to perform the full wavelet calculation at the end of the row/column/slice.

Our solution to this problem is to allow the user to select a border preservation option when generating low resolutions. When enabled, the edges of the data set are preserved in the lower resolution data sets. How this is done depends on how the lower resolutions are generated.

A.2 Decimation Border Preservation

Using decimation, the edges along the axes are always saved. Figure A-1a shows an example where the last point in each row is lost without border preservation. The only change that has to be made is to add an additional data point to the end of each row as shown in Figure A-1b. In this example, the last row is already included in the lower resolution, so no special action is required. In general, the last value along a dimension is already included if the dimension's size is $n2^r + 1$, where r is the resolution and n is some integer. At most, each dimension of the lower data set is increased by one. As a result, the size of the first resolution generated from the original data set of size $n \times m$, is always$(n+1)/2 \times (m+1)/2$ (using integer arithmetic), no matter the values of n and m.

Additional points retain their original geometry space positions. However, there are two options for placing the new points in the index space. One option, is to use the base coordinate system and place the points where they were sampled. This requires some previous knowledge, including the dimensions of the original data set and how border preservation is implemented. The other option is to place the points uniformly along with the rest of the low resolution data set. In this case, the additional points are moved to new positions outside of the original domain, preserving the uniform placement of low resolution points in the index space.

A.3 Wavelet Border Preservation

Preserving the edges with wavelets is not as simple. Because of the nature of the wavelet method, all sides of the resulting data set are reduced and require preservation. As shown in Figure A-2 all corner points must be saved and new data generated along each dimension in order to preserve the domain's size. This is achieved by applying the appropriate wavelet transformation to the first and last row and column of data (in the two-dimensions). In the case of three dimensions, both two and one dimensional wavelets would be used to capture the border data.

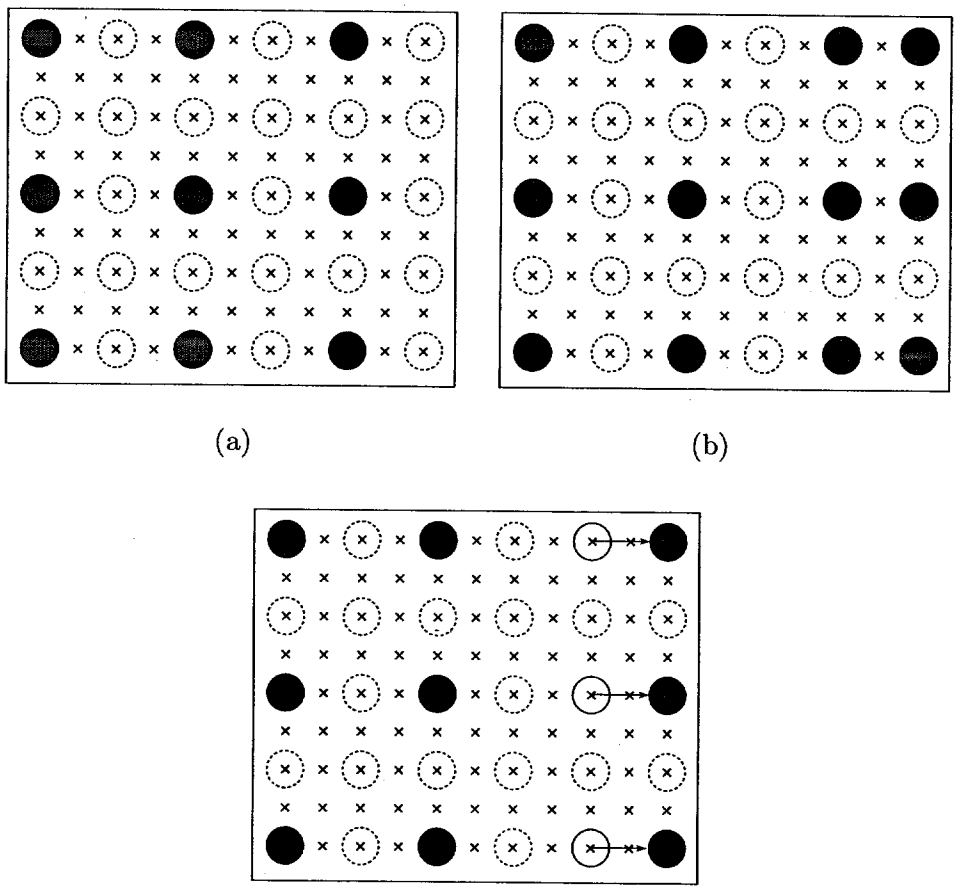


Figure A-1: First resolution data sets generated with uniform decimation. Red circles indicate points directly sampled from the original data set; dotted circles show the placement of unsampled points. (a) First resolution. (b) Points added to preserve the domain border. (c) Points shifted to preserve the domain size.

The resulting border data can be placed in the index space using two methods. The first is to locate the data at the points where it was calculated (see Figure A-2). This requires the application to know that the data set was generated with border preservation so the edge data can be placed in this non-uniform way. The alternative is to display the data uniformly, as if it had been generated without border preservation. To do this, the base coordinate system must be extended to allow for two additional rows and columns to be added to both ends of each dimension in the original resolution. Each data point is "shifted", from its original position to the new wavelet points. Data on the leading edges as well as data on the trailing edges of dimensions with an even number of points, are shifted by 3 points in the base resolution. Trailing edges of dimensions with an odd number of points will only need to be shifted by 1. Corners are shifted twice, once for each edge they lie on. The geometry space positions are calculated along with each value, and are not dependent on the index space position for the data point.

From an original resolution of $n \times m$, a first resolution data set generated with border preservation has a size of $(n/2 + 2) \times (m/2 + 2)$ regardless of the parity of n and m.

A.4 Gap Preservation for Adaptive Resolutions

In adaptive resolutions, neighboring regions of differing resolution can cause gaps in the domain when visualized. Just as in border preservation, extra data points can be introduced to close the spaces in between the subvolumes. How this is done depends on what data reduction method is used.

A.4.1 Gap Preservation in Decimation

In uniform decimation, points that are sampled from the original data set retain their original positions. When the interval of a single decimation step is a power of 2, every point contained in the lowest resolution is contained in all resolutions. This provides one solution to the gap problem. If subvolumes are selected, so that each starts on the boundary of the

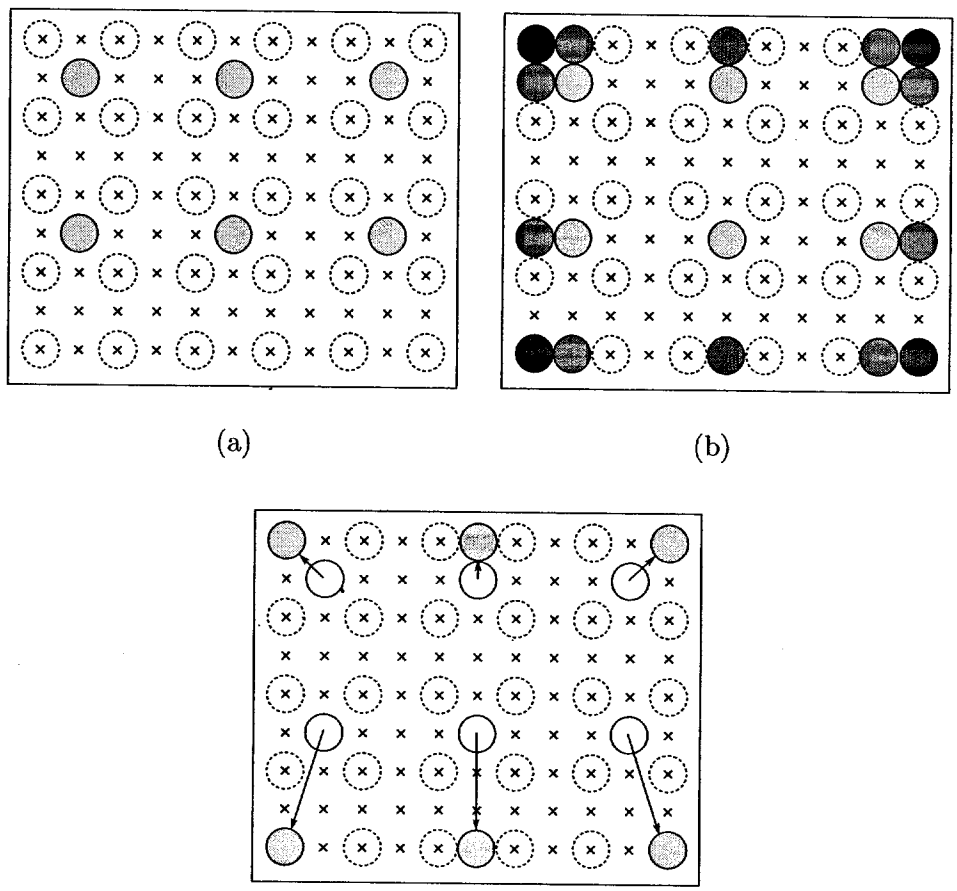


Figure A-2: First resolution data sets generated through wavelet decomposition. Red circles indicate points directly sampled from the original data set; yellow circles represent base resolution position of summary values; orange circles are summary values from one dimensional wavelets; dotted circles show the placement of unsampled points. (a) First resolution. (b) Points added to preserve the domain borders. (c) Points shifted to preserve the domain size.

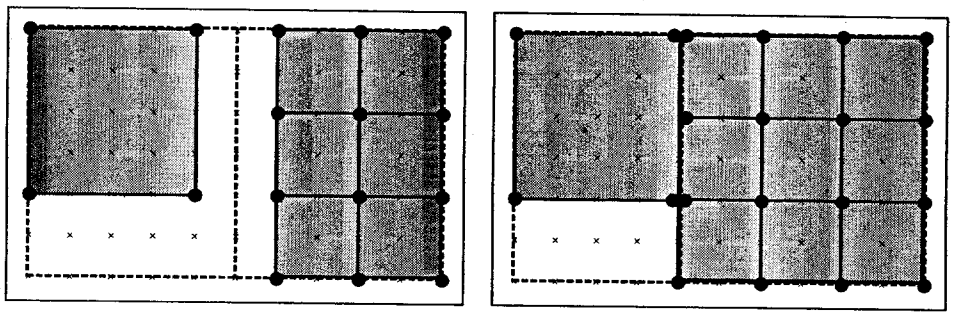


Figure A-3: (a) A simple adaptive resolution representation built from two resolutions. (b) Boundaries of each region fall on low resolution points, eliminating the interior gap.

lowest resolution, there are no interior gaps. In Figure A-3a, the 6×4 domain is divided in half, resulting in a gap between the two regions. However, if it is split into two regions (Figure A-3b), one 3×4, the other 4×4, there would be no gap, since the two resolutions share points on that edge. The only gaps left are on borders at the edge of the domain, which can be filled using border preservation.

A.4.2 Gap Preservation in Wavelets

Filling gaps in between different resolutions generated through wavelet decomposition is not as simple. In a wavelet multiresolution hierarchy, points are not shared between levels at all. Regions can be selected to minimize the size of the gaps, but cannot close them completely. To fully close them, additional data points can be added to the edge of one of the regions.

Figure A-4b shows a gap closed by extending the high resolution region (right). In this example, four new interpolated points are added, resulting in three new cells. To minimize the number of additional cells, gaps can be filled by extending the lower resolution at a boundary, to meet the edge of the higher resolution region (Figure A-4c).

A third option is to allow the data for each region to overlap (Figure A-4d). This is done by supplying data outside of the region's bounds in response to a data query for the region. Since this data already exists in the data set, there is no need for any interpolation.

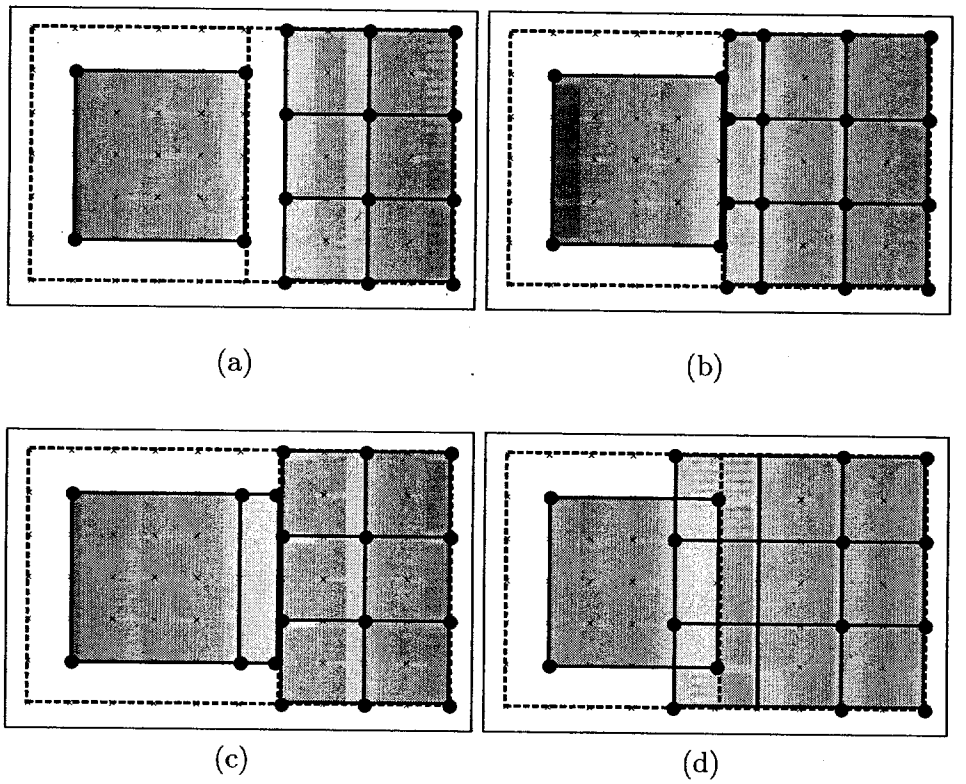


Figure A-4: (a) A simple adaptive resolution representation built from two resolutions. (b) Higher resolution is extended through interpolation to close the gap. (c) Low resolution is extended. (d) An extra column is fetched with the query for the second subdomain (no interpolated points).

Generally, areas of high resolution should be extended over areas of low resolution to avoid introducing error outside of the specified tolerance.

A.5 Implementation

Border preservation through point shifting is included as an option in the tool for building adaptive resolutions. This was chosen for its ease of implementation and because it does not require new low resolution data sets to be generated.

Gap preservation is implemented as an option for adaptive resolutions built with wavelet decomposition. The gaps between subdomains are filled by extending the bounds of each subdomain by 1 in each dimension and allowing them to overlap. Only data from the low

resolution data sets is used without any need to calculate values for intermediate points. Gap preservation for data sets generated with uniform decimation is not necessary because subvolumes are always divided along the points included in the lowest resolution data set.

Appendix B

USER'S GUIDE

This chapter describes the tools used to build the multiresolution and adaptive resolution representations described in this thesis.

B.1 MREBuilder

The multiresolution with error builder (MREBuilder) generates a multiresolution hierarchy with localized error information from a single, high resolution data source. The original data set is provided through a GRANITE File Descriptor Language (FDL) file or as a binary file if the `--binary` option is used. This tool works by reading in the minimum number of slices of the original data set needed to calculate the error for one slice at the lowest error resolution. For each resolution level, a low resolution version of the subset is generated and used to reconstruct the original resolution. Once the reconstruction for this subset is complete, error values are calculated for each error resolution and error function. Values for additional error functions and resolutions can be calculated easily without repeating the reconstruction process.

The low resolution data and error are written to the same directory as the original data set. The resolution levels and error function names are appended to the original file name to make the file names for the new data sets. For example, for file `timestep01` files `timestep01.d1`, `timestep01.d2`, and `timestep01.d3` are created for the lower resolution data sets and files `timestep01.d1.e2.maxabs`, `timestep01.d2.e3.maxabs`, and `timestep01.d3.e4.maxabs` are created for the error sets.

Finally, a Multiresolution Descriptor Language (MRDL) file is written along with the data and error files. This file describes the structure of the multiresolution hierarchy including the resolution levels, references to the data and error file descriptors, and the relationships between data and error sets (Figure B-1).

B.1.1 Usage

```
MREBuilder [OPTIONS] FILE
```

B.1.2 Options

Reducing Operators

`--uniform`

Use uniform decimation to build lower resolutions (default).

`--wavelet`

Use wavelet decomposition to build lower resolutions.

Resolutions

`-n NUMBER-OF-RESOLUTIONS`

The number of lower resolutions to generate. The default value is 1.

`-e NUMBER-OF-ERROR-RESOLUTIONS`

The number of error resolutions to generate. The default value is 1.

`-o ERROR-RESOLUTION-OFFSET`

The resolution offset between a low resolution data set and its first error set. The default value is 0.

Error Functions

`--maxabs`

Calculate the maximum absolute error for each region.

`--avgabs`

Calculate the average absolute error for each region.

`--stddev`

Calculate the standard deviation for each region.

`--snr`

Calculate the signal-to-noise ratio for each region.

Other

`--binary WIDTH HEIGHT DEPTH`

Generate an FDL file for the binary input file. This assumes the file contains single floating-point values and little endian byte order.

`--nofdl`

Suppresses FDL file output for the low resolutions and error sets.

B.2 ARBlocksWriter

This tool generates the subblocks and submeshes for an adaption resolution from a multiresolution data set with error and a user-specified error tolerance. The multiresolution data is read through a Multiresolution Descriptor Language (MRDL) file. This file stores the structure of the multiresolution data set and references to the FDL files for the data and error sets.

From this data set, an adaptive resolution tree is generated and pruned based on the error tolerance. A final pass is made to collapse children of equal resolution into the parent node (see Chapter 4). The resulting leaves are written out as a directory structure with the directory `ARBlocks/` as the root. This directory contains a subdirectory for each block (leaf) in the adaptive resolution. Each subdirectory contains a binary data file, and a grid file that describes the data point's location in base coordinates.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mr_descriptor PUBLIC "-//SDB//DTD//EN"
        "http://www.cs.unh.edu/~svohr/DTD/mrdl.dtd">
<mr_descriptor levels="2" offset="3" coords="uniform">
    <level id="0">
        <source type="data" resolution="0"
                desc="timestep01.fdl"/>
    </level>
    <level id="1">
        <source type="data" resolution="1"
                desc="timestep01.d1.fdl"/>
        <source type="error" resolution="4"
                desc="timestep01.d1.e4.maxabs.fdl"/>
    </level>
    <level id="2">
        <source type="data" resolution="2"
                desc="timestep01.d2.fdl"/>
        <source type="error" resolution="4"
                desc="timestep01.d2.e5.maxabs.fdl"/>
    </level>
    <level id="3">
        <source type="data" resolution="3"
                desc="timestep01.d3.fdl"/>
        <source type="error" resolution="4"
                desc="timestep01.d3.e6.maxabs.fdl"/>
    </level>
</mr_descriptor>
```

Figure B-1: A sample MRDL file with 4 levels at error resolution offset 3.

This tool also provides options for extending edges to match the original domain and filling in gaps between subvolumes of differing resolutions (generated through wavelet decomposition).

B.2.1 Usage

```
ARBlocksWriter [OPTIONS] IN-FILE
```

B.2.2 Options

`-e VALUE`

The error tolerance for the adaptive resolution. If unspecified, a "full" tree will be built.

`-b`

Shift the points along the edges to match the original domain.

`-g`

Fill gaps in the domain by allowing subdomains to overlap (wavelet only).

B.3 ar2silo

This tool takes the directory structure generated by `ARBlocksWriter` and the grid geometry file for the original resolution and builds a Silo file for the adaptive resolution. Each block is added as a quad mesh and variable with a multi-mesh and multi-variable added for all blocks.

B.3.1 Usage

```
ar2silo [OPTIONS] GEOMETRY-FILE BLOCKS-DIR
```

`-o FILENAME`

Set a name for the output file. If this option is not present, the filename `ar-grid.silo`

is used.

`-v`

Enables verbose mode. Prints each block as it is processed.