Space Efficient Fast Isosurface Extraction for Large Datasets

Udeepta D. Bordoloi, Han-Wei Shen
Department of Computer and Information Science
The Ohio State University
{bordoloi,hwshen}@cis.ohio-state.edu

Abstract

In this paper, we present a space efficient algorithm for speeding up isosurface extraction. Even though there exist algorithms that can achieve optimal search performance to identify isosurface cells, they prove impractical for large datasets due to a high storage overhead. With the dual goals of achieving fast isosurface extraction and simultaneously reducing the space requirement, we introduce an algorithm based on transform coding to compress the interval information of the cells in a dataset. Compression is achieved by first transforming the cell intervals (minima, maxima) into a form which allows more efficient compaction. It is followed by a dataset optimized non-uniform quantization stage. The compressed data is stored in a data structure that allows fast searches in the compression domain, thus eliminating the need to retrieve the original representation of intervals at run-time. The space requirement of our search data structure is the mandatory cost of storing every cell id once, plus an overhead for quantization information. The overhead is typically in the order of a few hundredths of the dataset size.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques.

Keywords: Isosurface, Compression, Transform Coding, Quantization

1 Introduction

Isosurfacing is one of the most popular methods for visually representing volumetric scalar fields. The effectiveness of isosurface visualization is, however, limited to a large extent by the interactivity of the visualization environment. The scope for interaction lies in two orthogonal components: isovalue (isosurface extraction phase) and view parameters (rendering phase). The usefulness of the visualization system is severely restricted if either the extraction or the rendering cannot be performed at interactive speeds.

Over the past few years, the sizes of the datasets being visualized using isosurfaces (among others techniques) have grown at an extraordinary rate. Consequently, achieving interactive speeds for the isosurface extraction phase has become progressively more and more challenging. Researchers have proposed a variety of isosurface-containing-cell[1] search techniques to expedite the process of isosurface extraction. These algorithms are motivated by the fact that given an isovalue, the volume needs to be searched for the cells that contain the isosurface. By pre-computing search-friendly data structures, these techniques reduce the time needed to search for those cells at run-time. Some techniques approach the problem as a search in geometric-space. Others, commonly known as value-space methods, search the space of intervals[2]. In this paper, we present a value-space algorithm.

A number of algorithms have been designed based on the concept of value-space. These algorithms achieve nearly optimal [Livnat et al. 1996] or optimal [Cignoni et al. 1997] speeds for the cell search phase. However, they suffer from one significant disadvantage: the storage requirement for the pre-computed search data structures. With very large datasets (such as the visible human dataset) becoming commonplace, the high storage overhead associated with these search structures is a serious deterrent to their use. For example, [Cignoni et al. 1997] states that the space requirement of Interval trees is four times the number of cells in the dataset. The Interval tree data structure for a 512^3 floating point dataset (512MB) will need more than 2036MB for storage. The large space complexity renders these techniques ([Giles and Haimes 1990][Shen and Johnson 1995][Livnat et al. 1996][Shen et al. 1996][Cignoni et al. 1997]) practically unusable without out-of-core modifications. Moreover, the algorithms are slowed down considerably because a large amount of time is spent on file I/O.

With the primary objective of fast isosurface extraction, we propose a compression-based solution intended to alleviate the problem of bloated search data structures. The extremal information (minimum and maximum) of the cells is compacted using a form of compression referred to as transform coding. We propose a computationally inexpensive transform of the conventional *[minimum, maximum]* representation of intervals[2] to a more compression friendly format. A traditional non-uniform quantizer is used to quantize the transformed data. We introduce a search algorithm to perform the isosurface search directly on the transformed representation. This eliminates the need for a decoding step to revert the data back to their original values. The search algorithm can be easily modified to do an incremental search, or to run out-of-core. We are able to reduce the size of the search data structures almost three-fold compared to those used by ISSUE and Interval trees. The compression technique presented provides a storage friendly yet efficient solution for large dataset isosurface extraction. The trade-off between storage requirements and the speed of the search process can be exploited to suit the available storage resources and the performance demands of the visualization environment.

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

[1] We use the term *cell* while referring to the smallest volumetric element in a three-dimensional grid. For regular grids, a cell represents the same entity as a voxel. For unstructured grids, a cell may be a tetrahedron, prism, or any other polyhedron. Our method can be used for datasets on either structured or unstructured grids.

[2] *Interval* here refers to the *[minimum, maximum]* interval of a cell.

2 Related Work

Since [Lorensen and Cline 1987] proposed the Marching Cubes algorithm for constructing isosurfaces in 1987, a number of techniques have been proposed to speed up the search for isosurface containing cells. Active list [Giles and Haimes 1990], Span Filter [Gallagher 1991], Sweeping Simplices [Shen and Johnson 1995], and Octrees [Wilhelm and Van Gelder 1992] are a few of the early methods. The first three are value-space based methods, while the ever popular octree is a geometric-space technique utilizing hierarchical spatial subdivision. [Itoh and Koyamada 1995] (extrema graphs) and [Bajaj et al. 1996] (seed cell set) use isosurface propagation techniques to avoid the need to search all the cells intersected by the isosurface. Propagation in unstructured grids needs adjacency information to be stored, which increases the storage. Below, we mention three value-space algorithms which are most related to our technique.

In 1996, [Livnat et al. 1996] introduced the span space representation for intervals in a near optimal algorithm (NOISE). The span space is a two-dimensional space of intervals with the x-axis and y-axis representing minima and maxima respectively. Each cell can be depicted as a point in the span space with the coordinates $(minimum, maximum)$. The span space is subdivided using a kd-tree, where each node divides the space into two partitions. The subdivision is alternated between a partitioning of the minima-axis and the maxima-axis at even and odd levels. The ISSUE algorithm by [Shen et al. 1996] employs a lattice-based subdivision of the span-space. Sequential and parallel algorithms are presented for performing a search over the lattice elements. [Cignoni et al. 1997] proposed an optimal search algorithm using Interval trees. Each node of the tree divides the intervals into three groups: the intervals whose maxima are less than the value of the node, those whose minima are greater than the node value, and the third set which contain the node value in between their extrema. The first group of intervals are passed onto the left child, the second to the right child, and the third group is put into two sorted lists associated with the node. Next, we discuss the storage requirements of NOISE, ISSUE and Interval trees.

Let us assume that there are N cells in the dataset, and the identity of each cell (cell id) is stored as a number that requires c bytes. Also, suppose that each data value requires d bytes. A pointerless kd-tree, as used in NOISE, stores the information {cell id, *minimum*, *maximum*} once for each cell. The space requirement is thus $(c+2d)N$. In ISSUE, all the lattice elements (except those intersected by the $minimum = maximum$ line) store two data structures. *Row* is a list of {cell id, *maximum*} pairs sorted by the cell maxima. The *Column* list comprises of {cell id, *minimum*} sorted by cell minima. Each cell in a lattice element contributes once to both *Row* and *Column* structures. So, the space needed is $(c+d)2N$, plus overhead. Each node of Interval trees stores two sorted lists: $\mathscr{AL}$ and $\mathscr{DR}$. $\mathscr{AL}$ is an ascending list of left extremes, *i.e.*, of {cell id, *minimum*} pairs, and $\mathscr{DR}$ is a descending list of right extremes, *i.e.*, of {cell id, *maximum*} pairs. Ignoring the tree overhead, the space needed is $(c+d)2N$. If cell ids are stored as 4-byte (one word) integers and data values as 4-byte floats, then the space requirement of NOISE, ISSUE, and Interval trees is respectively $3N$, $4N$ and $4N$ words. For a 512^3 floating point dataset (512MB), for instance, $N = 511^3$ and $4N$ words occupy 2036MB.

In the case of large datasets, which are common nowadays, the high storage requirement severely restricts the usability of these algorithms. This has prompted researchers to propose modifications so that large datasets can be used with these algorithms. [Cignoni et al. 1997] presents a 3D chess-board arrangement for regular grids to reduce the number of cells the interval tree stores. Cells are colored using a chess-board pattern, and only cells having black color are used to construct the interval tree. [Chiang and Silva 1997] proposed the first out-of core isosurfacing technique in the form of an I/O optimal implementation of the interval tree. Later, [Chiang et al. 1998] introduced a method to efficiently group individual cells into meta-cells. They construct an interval tree using the meta-cells instead of individual cells. Both the chess-board and the meta-cell techniques lower the space requirement by reducing the number of cells stored in the search data structures. Our algorithm achieves the same goal through efficient space utilization combined with compression of cell $[maximum, minimum]$ information. The compression method used is based on transform coding. If desired, the cell reduction techniques mentioned above (chess-board and/or meta-cell) can be incorporated into our algorithm to further decrease the search structure size. Let the effective number of cells (individual cells, or black cells in the chess-board pattern, or meta-cells) be N. In an uncompressed form, the {cell id, *minimum*, *maximum*} information requires $3N$ words. Using transform coding, we compress the {*minimum*, *maximum*} information to a few hundredths of N words. The total space requirement of our method is thus one and a few hundredths of N words, as opposed to $4N$ words in [Giles and Haimes 1990][Shen and Johnson 1995][Shen et al. 1996][Cignoni et al. 1997].

Transform coding is a well known data compression approach, and has an extensive body of literature. The basic principle utilized by transform coding is that multiple dimensions of vector data are often correlated to a lesser or higher degree. (If the input data is scalar, multiple samples are collected to form a vector.) The redundancy of data values (due to correlation) is exploited for compression by transforming the vector data and then quantizing each scalar dimension. The transformation allows a better compaction of the data compared to the untransformed values. The best compression ratios are achieved if the transformed data dimensions are not statistically correlated. [Hotelling 1933] presented the first transform to de-correlate discrete data in the method of *principal components*. Karhunen and Loéve derived the analogous transform for continuous functions, which is now popularly known as the K-L transform [Sayood 2000][Gray and Neuhoff 1998]. One of the most widely used transform coding applications today is the discrete cosine transform (DCT), which is a part of many image and video coding standards, *e.g.*, JPEG, MPEG etc. For a more detailed review of transform coding and quantization, the reader is referred to [Sayood 2000] and [Gray and Neuhoff 1998].

In the ensuing sections, we present our compression based algorithm for fast isosurface extraction.

3 Transform coding for intervals

A number of isosurface extraction algorithms have been developed to perform the search for cells in the value space, *i.e.*, the space of $[minimum, maximum]$ intervals of cells. The minima-maxima space, however, is not suitable for compression due to the high statistical dependence between the *minimum* and *maximum* values of cells (see figure 1). To reduce this dependence, we use a linear transform to transform this space into a new space (which we will refer to as the **UV**-space). This transformation is the first stage of our compression algorithm. Sections 3.1 and 3.2 discuss this step in greater detail. In the **UV**-space, each cell (or equivalently, each interval) is represented by its u- and v-coordinates. These coordinates are quantized using a dataset distribution optimized non-uniform quantizer. We use a companded quantizer, which simulates the non-uniform quantization process using a uniform quantizer. While choosing the output values of the quantizer, quantization errors are taken into account. This ensures that the isosurface search does not miss any cell that contains the isosurface. The quantization process is described in section 3.3. The compressed information (in the form of uv-coordinates) is then stored in a search friendly data structure, which is presented in section 4.1. At run-time, the search

algorithm finds the cells for isosurface extraction based on the supplied isovalue. The search process is explained in section 4.2.

In the following sections, we give some background on transform coding, followed by details of the transform and quantization phases of our algorithm.

3.1 Background

The central theme of transform coding is that the input data is modified, using a reversible transform, to another form which can be better quantized. The quantized data can then be converted back to the original form using the reverse transform. For the following discussion, we represent a multi-dimensional input data sample as the vector **x**. There are three stages in transform coding:

1. **Transform**: The input data **x** is transformed into **y** using a reversible transform **A**, where $\mathbf{y} = \mathbf{A}\mathbf{x}$. The transformation **A** is selected such that **y** has better compression characteristics than **x**, *i.e.*, given a fixed distortion, compressing **y** yields a smaller output than that of **x**. Or, given a fixed compression rate, **y** has lower distortion compared to **x**. The best compression results are achieved when the data dimensions are uncorrelated. This suggests that the ideal transform for compaction is the method of *principal components*. This step by itself does not result in any compaction of the data, which is achieved by the next two steps.

2. **Quantization**: The transformed data **y** is then quantized to a finite number of levels. Each dimension of the data can be quantized independently using different quantization strategies. The number of quantization levels depends on the desired amount of compaction. The statistics of the data **y** influence the design of the quantizer. For example, appropriate uniform or non-uniform quantizers can be chosen depending on input data properties and desired output statistics.

3. **Encoding**: The quantized data is then passed through a binary encoding stage (*e.g.*, Huffman or arithmetic coding). This results is the final compressed form of the data.

Constructing a compression scheme thus boils down to three tasks: finding an appropriate transform, designing quantizers based on the desired compression ratio and error limit constraints, and selecting a proper binary encoder. The data decoding process consists of inverting the effects of the first and third stages above. The second stage is lossy, and that information cannot be recovered. The compressed data is passed through a matching binary **decoder**, and then an **inverse transform** $\mathbf{A}^{-1}$ is applied to recover the data in the original form.

For the problem we are concerned with, the input data is a set of two-dimensional points which represent the [*minimum*, *maximum*] intervals of cells. In the rest of section 3, we propose a suitable transformation for the intervals, and then design a quantization scheme for the two transformed axes. Since our primary goal is fast isosurface extraction, we do not use any binary encoding stage. Such a stage would necessitate a decoder during the cell search phase, which would slow it down and defeat the primary purpose of this paper. However, if the situation so demands, a binary encoder can be easily applied as the third stage of encoding.

3.2 Transform

The best compression rates can be attained if we use a transformation which statistically de-correlates the minima and the maxima [Sayood 2000]. Hence, the ideal choice for a transformation is the method of principal components. However, it is very expensive to compute, specially for large datasets, which makes it a very impractical choice. Instead, we use a simpler transformation based on the following observation: *it is usual for the minima and maxima of the cells to be highly correlated.* That is, cells with higher maxima tend to have higher minima and vice versa. Figure 1 shows a histogram plot of the difference between *maximum* and *minimum* values in the visible woman dataset. As can be expected, the vast majority of cells have a very small difference between their maxima and minima.

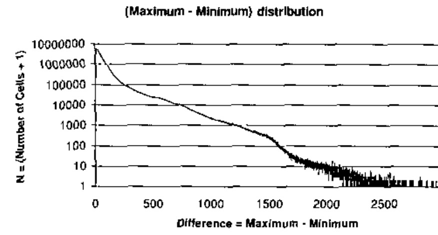

Figure 1: A histogram plot of the difference between *maximum* and *minimum* values in the visible woman dataset. The x-axis represents the difference, and the y-axis shows the number of cells which have that difference. The y-axis is shown in a $\log_{10}$ scale. In this dataset, the largest difference is 2978, but 90% of cells have a difference less than 163.

Consider a two-dimensional space in which the x-axis represents the cell minima and the y-axis represents the maxima (span space in [Livnat et al. 1996]). Since the cells tend to distribute themselves along the *minimum* = *maximum* line, the principal component of any dataset will have an orientation close to the *minimum* = *maximum* line. So, instead of the exact principal component transformation, we use a transformation to the 45° line. (Note that the transformation can be interpreted as rotation of the coordinate frame). Each interval is represented as a vector

$$\mathbf{x} = \begin{bmatrix} minimum \\ maximum \end{bmatrix} \tag{1}$$

The transformation is given by

$$\mathbf{A} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \tag{2}$$

where $\theta = 45°$. After the transformation, each interval is represented by the vector

$$\mathbf{y} = \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{A}\mathbf{x} \tag{3}$$

Or,

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} minimum \\ maximum \end{bmatrix} \tag{4}$$

We reduce the computational expense of the transform by removing the multiplication present in equation (4). This can be viewed as a scaling of both components of **y**, or equivalently as the use of a different system of units. Hence, this does not affect the compression results. Defining

$$\mathbf{y} = \mathbf{B}\mathbf{x} = (\sqrt{2}\mathbf{A})\mathbf{x} \tag{5}$$

we get

$$u = maximum + minimum \tag{6}$$

$$v = maximum - minimum \tag{7}$$

Each cell is represented as a point with coordinates (u, v) in the u-v frame, which is obtained by a counter-clockwise rotation of the

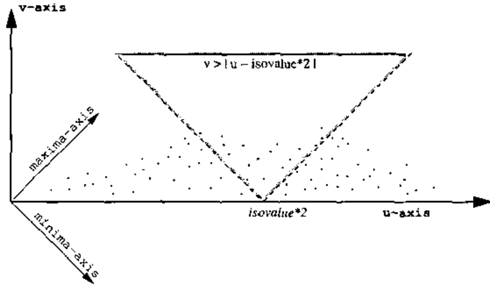

Figure 2: **UV**-Space. The **UV**-space is a two-dimensional space of intervals. Each cell is represented as a point with coordinates (u, v) defined by the equations (6) and (7). The u-v frame is obtained by a counter-clockwise rotation of the original *min-max* frame by 45°, followed by a scaling by $\sqrt{2}$. The isosurface passes through the cells in the shaded region.

original *min-max* frame by 45°, followed by a scaling with $\sqrt{2}$ (figure 2). We will refer to the two-dimensional space represented by the u-v frame as the **UV**-Space. Note that, u can be thought of as twice the mid-point of the interval, and v is the range of the interval.

Using the equations (6) and (7), the interval *minimum* and *maximum* can be expressed as

$$minimum = (u-v)/2 \quad (8)$$
$$maximum = (u+v)/2 \quad (9)$$

The cells which contain the isosurface satisfy the following condition:

$$minimum < isovalue < maximum \quad (10)$$

For simplicity, we have assumed *minimum* $\neq$ *isovalue* $\neq$ *maximum*. From equations (8), (9) and (10), we can derive the following condition for a cell which intersects the isosurface

$$v > | u - isovalue \times 2 | \quad (11)$$

In figure 2, any cell lying inside the shaded region (defined by equation (11)) will be intersected by the isosurface.

In addition to permitting better compression rates, the transformation given by equations (6) and (7) also has other advantages. First, it requires very little computation in the form of a couple of additive operations. An inverse transformation is not needed: the isosurface test can be done in the transform domain using equation (11). Moreover, the transformed space lends itself to a simple and efficient search data structure, which we will present in section 4. As will be evident, we will need to store only one sorted list of cells, as opposed to two sorted lists in most algorithms ([Giles and Haimes 1990][Shen and Johnson 1995][Shen et al. 1996][Cignoni et al. 1997]).

3.3 Quantization

Quantization of the **UV**-space is performed in two phases: first, the u-axis is quantized, followed by a quantization of the v-axis. For both axes, we use data distribution optimized non-uniform quantization.

3.3.1 Companded Quantization

After the transformations given by equations (6) and (7), let the minimum u value for the dataset be u_L, and the maximum u value be u_R. We want to quantize the range $[u_L, u_R]$ into M intervals, where M is input by the user. The design of the quantizer involves deciding the following two sets of values:

- *Decision Boundaries*: The $M+1$ endpoints $\{b_i\}_{i=0}^{M}$ of the M intervals. We already have $b_0 = u_L$, and $b_M = u_R$.
- *Reconstruction Levels*: The M representative values $\{r_i\}_{i=1}^{M}$ for each interval.

The quantizer function, $Q(\cdot)$, is given by

$$Q(u) = r_i \qquad \text{iff} \quad b_{i-1} < u \leq b_i \quad (12)$$

Since the distribution of cells in along the u-axis can be (and usually is) non-uniform, we will use a non-uniform quantization strategy. Specifically, we will use an approach called Companded Quantization [Gray and Neuhoff 1998][Sayood 2000], which simulates a *distribution optimized* non-uniform quantizer. A compander has three stages:

1. *Compressor*: The input values (u coordinates of cells) are mapped into another value (say, u') such that the output (u') is uniformly distributed. The regions of the input which have high density are stretched, while regions with low density are compressed. The mapping conserves the ordering of the input values, *i.e.*, if $u_i < u_j$, then $u'_i < u'_j$. The concept is the same as that used in image equalization.
2. *Uniform Quantizer*: The output of the compressor stage (u') is quantized into M levels using a uniform quantizer. The decision boundaries of this quantizer are $\{b'_i\}_{i=0}^{M}$, and the reconstruction values are $\{r'_i\}_{i=1}^{M}$.
3. *Expander*: The quantized u' values are mapped back to the u-axis using an expander function, which inverts the warping introduced by the compressor function. The compander decision bounds $\{b_i\}_{i=0}^{M}$ are derived from $\{b'_i\}_{i=0}^{M}$, and the reconstruction levels $\{r_i\}_{i=1}^{M}$ are obtained from $\{r'_i\}_{i=1}^{M}$.

3.3.2 Quantization of u-axis

For the first phase, the user specifies the number of quantization intervals, M, of the u-coordinates. We implement the compressor stage by sorting the cells by their u-coordinates. If the u-values of two cells are equal, we break the tie using cell ids. The position of a cell in the sorted sequence is used as its u' value for the uniform quantizer. The first $n_M = N/M$ cells are quantized into the first interval, the next n_M cells in the second interval and so on. The decision boundaries, $\{b'_i\}_{i=0}^{M}$, of the uniform quantizer are the sequence positions of the extreme (the first, and the last) cells of the intervals. The expander stage involves mapping the $\{b'_i\}_{i=0}^{M}$ values to the u-axis using an inverse of the compressor stage. Let the u-value of the jth cell (in the sorted sequence) be u_j, and let $\eta = n_M$. Then the compander decision boundaries, $\{b_i\}_{i=0}^{M}$, are defined as

$$\begin{array}{rcl} b_0 & = & u_L \\ b_1 & = & (u_\eta + u_{\eta+1})/2 \\ b_2 & = & (u_{2\eta} + u_{2\eta+1})/2 \\ \vdots & \vdots & \vdots \\ b_M & = & u_R \end{array} \quad (13)$$

We have assumed that $\{u_{i \cdot \eta} \neq u_{i \cdot \eta+1}\}_{i=1}^{M-1}$. If that does not hold, we take b_i as the average of the u-values of the next two satisfying cells. Figure 3 shows the quantization of u-coordinates with $M = 11$. The vertical lines at the decision boundaries $\{b_i\}_{i=0}^{11}$ divide the

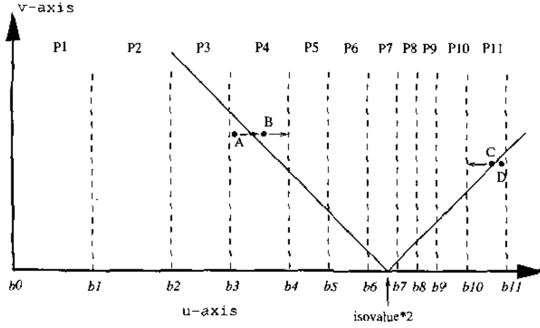

Figure 3: Quantization of u-axis. The u-axis is divided into $M = 11$ levels. The *decision boundaries* $\{b_i\}_{i=0}^{11}$ are given by equation (13). For the given isovalue, the *reconstruction levels* are given by equation (14): $r_1 = b_1, ..., r_6 = b_6, r_8 = b_7, ..., r_{11} = b_{10}$. r_7 is not used by the search algorithm. The $M = 11$ partitions of the UV-space will be referred to as *U-partitions*.

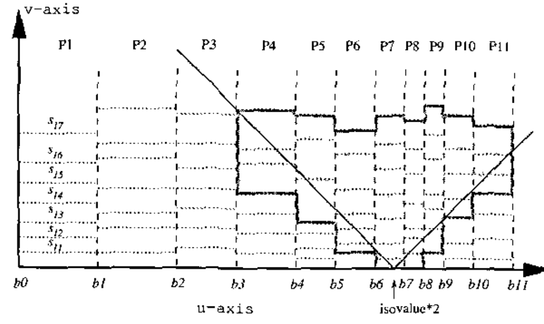

Figure 4: Quantization of v-axis. After the u-axis has been quantized using $M = 11$, the v-axis is quantized separately into $L = 7$ levels for each U-partition $\{P_i\}_{i=1}^{11}$. The v-axis *reconstruction levels*, which are also the *decision boundaries*, are shown as the horizontal lines, and the values are calculated from equation (15). The $L = 7$ intervals of each U-partition will be referred to as *UV-partitions*.

UV-space into $M = 11$ partitions $\{P_i\}_{i=1}^{11}$, which we will refer to as the *U-partitions*.

Unlike usual quantization procedures, the definition of the reconstruction values $\{r_i\}_{i=1}^{M}$ is deferred till run-time. To avoid holes in the isosurface due to quantization errors, we need to incorporate the isovalue into the assignment of $\{r_i\}_{i=1}^{M}$. Consider the cells A and B in the U-partition P_4 in figure 3, where the value $u_{iso} = 2 \times isovalue$ lies in U-partition P_7. Both will have the same quantized u-coordinate r_4, which will be used at run-time for the isosurface test in equation (11). If cell B fails the test, the resulting isosurface will have a hole in it. To prevent any potential isosurface cell from failing the test, we have to ensure that the right-hand side of the inequality ($v > | u - isovalue \times 2 |$) does not increase as a result of quantization. Hence, we choose the reconstruction level to be the greatest u-coordinate any cell in partition P_4 can take. This happens to be the right decision boundary of P_4, and so we take r_4 to be equal to b_4. For the same reasons, cells C and D in partition P_{11} are assigned the reconstruction value $r_{11} = b_{10}$. Note that cells A and D will satisfy equation (11) and will be sent to the geometry extraction phase, which will simply ignore them. For the partition P_7, which contains the value u_{iso}, all the cells are presumed to have passed the test. We define the reconstruction levels in terms of the stored decision boundaries and the given *isovalue* using the following formula: assuming $b_{iso-1} < isovalue \times 2 \leq b_{iso}$

$$\begin{array}{rcl} r_1 & = & b_1 \\ \vdots & \vdots & \vdots \\ r_{iso-1} & = & b_{iso-1} \\ r_{iso} & = & \text{not required} \\ r_{iso+1} & = & b_{iso} \\ \vdots & \vdots & \vdots \\ r_M & = & b_{M-1} \end{array} \quad (14)$$

3.3.3 Quantization of v-axis

After the quantization of the u-axis, we proceed to the second phase of our algorithm. We quantize the v-axis in each partition $\{P_i\}_{i=1}^{M}$ of the UV-space separately. The quantization strategy is similar to that used for the u-coordinates. The user specifies the number of quantization levels, L, for each U-partition. The following actions are then performed for each partition P_i ($i = 1...M$). The cells are initially sorted by their v-values, breaking ties by cell ids. The first $n_L = n_M/L$ cells are put in the first interval, the next n_L cells in the second interval and so on. Unlike the quantization stage of u-axis, we do not prevent cells with the same v-values from being put into different intervals. We do so to ensure that each interval contains the same number (n_L) of cells. As a result, we do not have to explicitly store the number of cells in each interval in our data structure.

In the previous discussion on quantizing the u-axis, we argued the need to prevent quantization errors which might result in holes due to isosurface-containing cells being indicated otherwise. In the isosurface test ($v > | u - isovalue \times 2 |$, equation (11)), this translates to the requirement that the v-value should not decrease after quantization. Accordingly, for each interval, the highest v-value of its member cells is used as the reconstruction level for that interval. Let the v-value of the kth cell (in the sorted sequence) in the U-partition P_i be v_{ik}. Then, the reconstruction values, $\{s_{il}\}_{l=1}^{L}$ are given by

$$\begin{array}{rcl} s_{i1} & = & v_{i(n_L)} \\ s_{i2} & = & v_{i(2n_L)} \\ \vdots & \vdots & \vdots \\ s_{iL} & = & v_{i(Ln_L)} \end{array} \quad (15)$$

Figure 4 shows the quantization of v-coordinates after the u-axis has been quantized (with $M = 11$, as shown in figure 3). Each U-partition has been further divided by $L (= 7)$ horizontal lines, which represent the reconstruction levels (also the decision boundaries) of the v-values within the U-partition. We will call the resulting rectangular regions *UV-partitions*. The UV-partitions which are to the left of the U-partition P_7 are represented by the uv-coordinates of their top-right corners. Similarly, those to the right of P_7 are represented by the uv-values of their top-left corners. For the given isovalue, the shaded UV-partitions pass the isosurface test as their representative corners satisfy equation (11).

4 Search Algorithm

Following the transform coding steps outlined in the previous section, we construct data structures which store the information of the UV-space in a compressed form (section 4.1). These can then be used for fast isosurface extraction searches (section 4.2).

4.1 Data Structures

The preprocessing stage of our algorithm consists of the transformation and quantization steps that have been mentioned in sections 3.2 and 3.3 respectively. The results of the preprocessing stage are stored in appropriate data structures that enable a fast run time search for isosurface containing cells. The information that needs to be stored is: the user-specified quantization parameters M and L, the reconstruction levels for the u- and v-axes, and the cell ids in each UV-partition. We use the data structures given below to store that information:

1. *U-Array*: The decision boundaries $\{b_i\}_{i=0}^{M}$ for the u-axis, given by equation (13). These values are required at run time to derive the reconstruction levels for u-coordinates according to equation (14). The storage required is the space for $M+1$ values.

2. *V-Array*: A two-dimensional array $\{d_{ij}\}_{i=1,j=1}^{M,L}$ with each element storing the v-axis reconstruction levels of the corresponding UV-partition given by equation (15). For example, d_{ij} stores the decision boundary of the jth UV-partition of the ith U-partition. This needs a storage of ML values.

3. *ID-Array*: A two-dimensional array $\{A_{ij}\}_{i=1,j=1}^{M,L}$ with each element storing the ids of cells in the corresponding UV-partition. $A(i,j)$ stores the cells within the jth UV-partition of the ith U-partition. The storage needed is that for N cell ids.

The total storage requirement is the space needed for N cell ids and $ML+M+1$ quantization levels, where $ML+M+1$ is typically of the order of a hundredth of N. This offers significant space reduction compared to most algorithms ([Giles and Haimes 1990][Shen and Johnson 1995][Shen et al. 1996][Cignoni et al. 1997]), which store $2N$ cell ids and $2N$ min-max values. During preprocessing, the three arrays are filled simultaneously through the quantization process described in section 3.3. To recap, each cell is transformed to uv-coordinates using equations (6) and (7). They are then sorted by their u-coordinates and the quantization interval endpoints $\{b_i\}_{i=0}^{M}$ derived using equation (13). The cells are then grouped into M U-partitions. The cells in each U-partition are now sorted by their v-values. For each U-partition i, the V-array elements (v-axis decision bounds) are filled in according to equation (15). Simultaneously, ids of cells in each UV-partition are stored in the ID-Array.

4.2 Search

Given an isovalue, the search for isosurface containing cells over the **UV**-space can be decomposed into separate searches over each U-partition. For a given U-partition, the search can be thought of as a search for satisfying UV-partitions (because all the cells within a given UV-partition have the same quantized uv-values). The U-partition is traversed in order of decreasing v-coordinates, beginning with the topmost UV-partition (the one with highest v-value). The reconstruction values of the UV-partition are read from the U-Array and the V-Array, and tested in equation (11). If the UV-partition satisfies the isosurface test, all the cells in the corresponding ID-Array position are selected for geometry extraction, and the search moves to the next UV-partition (the one below). When a UV-partition is reached whose uv-coordinates fail equation (11), the traversal for the current U-partition is terminated and the another U-partition is taken up for traversal. The search is complete when all the U-partitions have been individually searched.

Incremental Search

If the isovalue is changed by a small amount from the previous isovalue, it is advantageous to do an incremental update to the results of the previous search. We assume that the previous isovalue search results for each U-partition are stored. For each U-partition, we also need to remember the position of last UV-partition accessed before the traversal was terminated. Let the previous isovalue be iso_p. Without any loss of generality, let us assume that the isovalue has increased to a new value iso_n. Let the corresponding u-axis points be $u_p = 2 \times iso_p$ and $u_n = 2 \times iso_n$ respectively. Then the addition of new cells and removal of cells no longer intersecting the isosurface are handled as follows:

1. *Addition*: New cells will be added to U-partitions that are to the right of $u_{mid} = (u_p + u_n)/2$. For these U-partitions, we start an incremental search from the previous terminating UV-partition. The current traversal is continued till a UV-partition is reached which does not satisfy the isosurface condition (equation (11)). The cells of the newly traversed UV-partitions are added to the isosurface extraction list.

2. *Removal*: For U-partitions to the left of u_{mid}, we will need to potentially remove cells which were selected for isosurfacing for the previous isovalue. Each U-partition is traversed upwards (towards increasing v-values), starting from the terminating UV-partition of the previous traversal. The upward traversal is stopped when a UV-partition is reached which satisfies the isosurface test. The UV-partitions encountered during this reverse traversal no longer contain the isosurface and are removed.

The U-partition which contains u_{mid} can belongs to the addition category if the previous terminating UV-partition satisfies the isovalue. Otherwise, it is in the removal set. As in any incremental update search, this is more beneficial in case of small datasets, for which the intermediate results can be stored in main memory.

4.3 Errors

The quantization of the **UV**-space introduces errors which may result in false conclusions for some cells in the isosurface test (equation (11)). We have designed our quantizer (section 3.3) such that the search does not miss any cell that contains the isosurface. Instead, some cells that do not truly intersect the isosurface will satisfy equation (11). The errors are the combined effect of u-value quantization error and the v-axis quantization error. Below, we give an empirical discussion on the average effect of the u-axis quantization on the number of such erroneous cells. For this discussion, we first assume that the v-values are not quantized. Later, we will extend the error analysis to include the v-coordinate quantization.

Consider the U-partition P_i in figure 5(a), which is to the left of $u_{iso} = 2 \times isovalue$. In other words, $b_i < 2 \times isovalue$. Due to quantization of u-coordinates, all the cells within the shaded triangular region will satisfy the isosurface test, and will constitute the error for this U-partition. Each U-partition that is searched will contribute a similar group of erroneous cells. It should be noted that if the top-most UV-partition of a U-partition fails the isosurface test, it will not be traversed at all and hence will not contribute any error. For instance, in figure 4, the U-partitions P_1, P_2 and P_3 will not have any error since the topmost UV-partitions lie outside the isosurface region. In practice, the dynamic range of u-values is usually much higher than the spread of v-values. As a result, a large number of U-partitions will not be traversed and so will not contribute any error. For this discussion, we assume that on an average, a fraction h of the total number M of U-partitions is traversed. Let the average width of a U-partition be u_{ave}, and the mean concentration

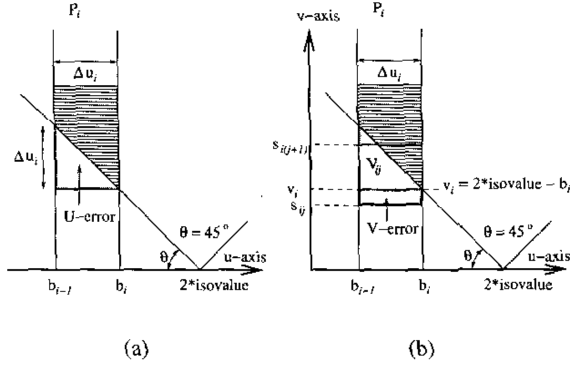

Figure 5: Quantization Errors. All the cells in the shaded triangular region in figure (a) satisfy the isosurface test and contribute to the error due to quantization of u-axis. The error that is added to this by v-axis quantization is shown in figure (b).

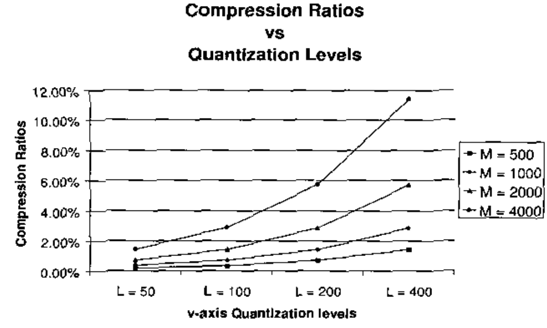

Figure 6: Effect of Quantization parameters M and L on data structure size.

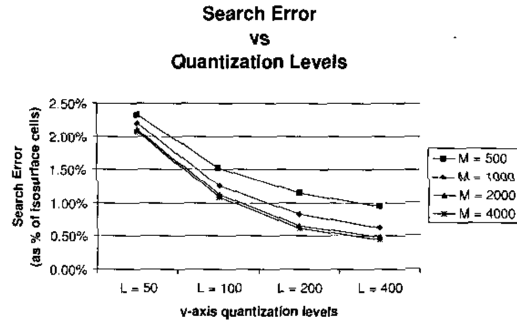

Figure 7: Effect of Quantization parameters M and L on search error for the MR-brain (isovalue = 1070.5, number of isosurface cells = 4352196).

of cells be c_{ave}. Then, on an average, each U-partition will contribute $(u_{ave}^2/2)c_{ave}$ false cells. If the u-value limits for the dataset are $[u_L, u_R]$, then the average number of cells which falsely satisfy equation (11) is

$$\text{Average U-Error} = hM(\frac{u_{ave}^2}{2} \cdot c_{ave}) = \frac{(u_R - u_L)^2 h c_{ave}}{2M} \quad (16)$$

Next, the additional effect of v-coordinate quantization is considered. In figure 5(b), all the cells in the UV-partition V_{ij} have the same uv-coordinates $(b_i, s_{i(j+1)})$, and thus all satisfy equation (11). The triangular region contains cells which incorrectly satisfy the isosurface test due to u-axis quantization errors. The error added by v-axis quantization are those cells in the UV-partition V_{ij} whose v-coordinates are less than $v_i = 2 \times isovalue - b_i$. If the total number of cells in the dataset is N, and M and L are the number of quantization levels for u- and v-axes respectively, then each UV-partition has $n_L = N/ML$ cells. On an average, the total number of erroneous cells due to v-coordinate quantization is

$$\text{Average V-Error} = hM \cdot \frac{n_L}{2} = \frac{hN}{2L} \quad (17)$$

5 Results and Discussion

In this section, we first discuss the effect of the quantization parameters M and L on the size and search efficiency of the search data structures. We then present out-of-core results from our algorithm and also compare the performance with that of the interval tree. We have tested our algorithm on the UNC MR-brain dataset ($256 \times 256 \times 109$ 2-byte integer), a Rayleigh-Taylor hydrodynamic instability dataset (256^3 floating-point) which we will refer to as Rage256, and the visible woman dataset ($512 \times 512 \times 1728$ 2-byte integer).

5.1 Compression and Errors

We have mentioned before that either the meta-cell technique [Chiang et al. 1998] or the chess-board method [Cignoni et al. 1997] can be used with our algorithm. For the following discussion, we will denote the number of effective cells (single cells, meta-cells, or black cells in the chess-board pattern) by N. The space requirement of our search data structure is the storage for N cell ids (ID-Array) and $ML + M + 1$ quantization levels (U-Array and V-Array). Since we are not compressing the cell ids, the space required to store the ID-Array will remain constant for all quantization parameters. We present the compression results as the ratio of the size of the U-Array and V-Array to the space required for storing the min-max values for every cell. Figure 6 shows the compression ratios for the MR-brain dataset. Interval trees and ISSUE data structures store $2N$ cell ids and N min-max pairs. Compared to these, the storage required by our search data structure is 37.1% for MR-brain, 27.4% for Rage256, and 33.4% for visible woman dataset for a $(M = 4000, L = 400)$ quantization.

Figure 7 shows the variation of error with L and M. The error is due to cells which are selected by the search algorithm but do not contain the isosurface. Please note that there is no error in the isosurface itself. The error is defined as the ratio of the erroneous cells to the number of isosurface containing cells. As expected, the error decreases with increase in both L and M. Note that the rate of decrease falls as L or M get larger. Keeping in mind the trade-off between search and space efficiencies, users can choose an (M,L) combination suitable for their requirements. For instance, the very little difference between performance of the $M = 2000$ and $M = 4000$ graphs may not justify the associated increase in storage space. Table 1 gives the preprocessing, search and extraction times for the MR-brain dataset for a subset of quantization parameters from figures 6 and 7.

M,L	Comp-ression	Error	Pre-process	Search	Extract
500, 50	0.18%	2.33%	11.03s	0.04s	7.48s
2000, 200	2.86%	0.66%	11.40s	0.03s	7.44s
4000, 400	11.42%	0.44%	11.50s	0.03s	7.43s

Table 1: Search and space efficiency trade-off. Processing times on a 600MHz PIII for different (M,L) combinations are shown (isovalue = 1070.5). The extraction time for a zero search error is 7.41s. The associated compression and errors are shown in figures 6 and 7.

Isovalue	Cells	Error	Search	Extraction
600.5	2,066,710	4.39%	0.05s	13.0s
1100.5	4,433,023	4.28%	0.12s	27.7s
1400.5	809,193	9.47%	0.04s	6.1s

Table 2: Search and extraction times for the visible woman dataset using a compression of 3.27% of the min-max values. The size of the search data structure is 34.5% of the size of the ISSUE/Interval-Tree data structures. $2 \times 2 \times 2$ meta-cells are used while constructing the search data structures. The number of isosurface containing meta-cells are given, along with the error introduced by quantization.

5.2 Performance

In case of large datasets, the search data structures may not fit into main memory and out-of-core techniques have to be implemented. Because we store the min-max information and the cell ids in separate data structures, we do not need to modify our search algorithm for large datasets. Only the U-Array and the V-Array need to be kept in-core. During the search phase, the V-Array is scanned as described in the search algorithm (sec.4.2). If the uv-coordinates stored at a V-Array position pass the isosurface test, the corresponding ID-Array entry is read from the disk and the cells passed to the extraction stage. Table 2 shows the search and extraction times for the visible woman dataset. For this experiment, we have used a $2 \times 2 \times 2$ meta-cell for constructing our data structure. The error (number of meta-cells selected due to quantization error) is given as a percentage of the isosurface meta-cells, given in the second column. The data-structure I/O times are included in the extraction times. The compression ratio of min-max information is 3.27% for the data structures used. The size of the search data structure is 34.5% of the size of the ISSUE/Interval-Tree data structures.

Table 3 compares the size and performance of our algorithm to an in-core interval tree implementation on a MIPS R10000 Processor. We present results for a floating-point MR-brain dataset and the Rage256 dataset for both methods. The interval tree search performs marginally better than the search using compressed min-max values. The search data structures of our algorithm are smaller by a factor of four or more compared to the interval tree.

6 Conclusion and Future Work

We have presented a data structure for speeding up isosurface extraction using transform coding techniques. Significant reduction is achieved in terms of the space requirement of the search structures, without compromising the search speed. In the future, we want to extend the compression to cell ids to further reduce the size of the search structure, and to extend the algorithm to time-varying data.

Dataset	Search Method	Search structure size	Search Time
MR-brain	QS (2.5%)	20.8MB	0.18s
MR-brain	I-Tree	93.8MB	0.13s
Rage256	QS (1.3%)	49.0MB	0.09s
Rage256	I-Tree	221.4MB	0.07s

Table 3: Comparison of search times for the quantized search (QS) and the interval tree (I-Tree). The compression ratios for the min-max data are given in parentheses.

Acknowledgement

This research was supported in part by NSF grant ACR 0118915, NASA grant NCC-1261, Ameritech Faculty Fellowship and Ohio State Seed Grant. We thank the anonymous reviewers for their helpful comments.

References

BAJAJ, C. L., PASCUCCI, V., AND SCHIKORE, D. R. 1996. Fast isocontouring for improved interactivity. In *1996 Symposium for Volume Visualization*, IEEE Computer Society Press, Los Alamitos, CA, 39–46.

CHIANG, Y.-J., AND SILVA, C. T. 1997. I/O optimal isosurface extraction. In *Proceedings of Visualization '97*, 293–300.

CHIANG, Y.-J., SILVA, C. T., AND SCHROEDER, W. J. 1998. Interactive out-of-core isosurface extraction. In *Proceedings of Visualization '98*, 293–300.

CIGNONI, P., MARINO, P., MONTANI, E., PUPPO, E., AND SCOPIGNO, R. 1997. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics 3*, 2, 158–170.

GALLAGHER, R. S. 1991. Span filter: An optimization scheme for volume visualization of large finite element models. In *Proceedings of Visualization '91*, 68–75.

GILES, M., AND HAIMES, R. 1990. Advanced interactive visualization for CFD. *Computing Systems in Engineering 1*, 1, 51–62.

GRAY, R. M., AND NEUHOFF, D. L. 1998. Quantization. *IEEE Transactions on Information Theory 44*, 6, 2325–2383.

HOTELLING, H. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psycology 24*, 417–441,498–520.

ITOH, T., AND KOYAMADA, K. 1995. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics 1*, 4, 319–327.

LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. 1996. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (March).

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics 21*, 4 (July), 163–169.

SAYOOD, K. 2000. *Introduction to Data Compression*. Morgan Kaufmann Publishers, Inc.

SHEN, H.-W., AND JOHNSON, C. R. 1995. Sweeping simplices: A fast isosurface extraction algorithm for unstructured grids. In *Proceedings of Visualization '95*, 143–151.

SHEN, H.-W., HANSEN, C. D., LIVNAT, Y., AND JOHNSON, C. R. 1996. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of Visualization '96*, 287–294.

WILHELM, J., AND VAN GELDER, A. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 3 (July), 201–227.