EG UK Theory and Practice of Computer Graphics (2010)
John Collomosse, Ian Grimstead (Editors)

Interval Based Data Structure Optimization

B. Duffy[1] and H. Carr[2]

[1]University College Dublin, Dublin, Ireland
[2]University of Leeds, Leeds, England

Abstract

Isosurface extraction is a widely exploited visualization technique for volumetric data on all manner of grid representation. The basic technique is often used to explore and measure many properties of data sets of ever increasing size. Therefore, data structures and algorithms that facilitate interactive exploration and fast processing of isosurfaces of large data sets is of paramount importance. While many optimal methods have been proposed to accelerate isosurface extraction, many of these algorithms have limitations with regards to storage costs and data quantization. In some cases these limitations preclude their practical application. We present a very simple clustering and volume compression technique based on observations in the span space and show that applying this technique to existing methods can reduce their storage cost. We show results for real data validating our technique.

Categories and Subject Descriptors (according to ACM CCS): Clustering, Span Space, Quantization

1. Introduction

Many disciplines such as engineering, science and medicine produce large quantities of 3D volumetric data sets. In particular scalar fields of single observations of natural processes and physical phenomena are routinely acquired from scanning equipment and the size of these data sets increase as storage and technology improves. Scientific visualization continues to play an important role in understanding, exploring and measuring this data and a wide variety of visualization techniques have been reported for this purpose. One specific technique that is widely exploited is isosurface extraction, where surfaces of the form $\{x : f(x) = v\}$ are used for boundary extraction or direct visualization. The surfaces extracted are intermediate geometric approximations of features in the data and many methods have been proposed to facilitate accelerated and even optimal extraction of isosurface. Some methods are limited to quantized data and some methods are limited by their data structure.

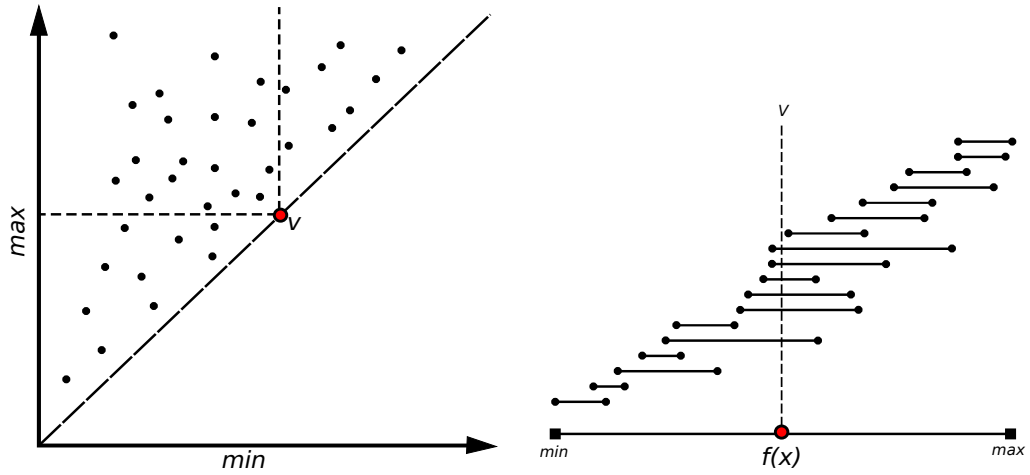
We describe a very simple volume clustering and compression technique based on observations in the span space that can be boot-strapped to an interval based isosurface acceleration technique to increase the performance of the basic algorithm or data structure at an extra pre-processing cost.

The rest of the paper is laid out as follows. Section 2 is a review of the relevant related work in the area of isosurface extraction. Section 3 is a review of the span space data representation and a number of observation about span space. Section 4 describes our technique for representing the volumetric data sets. Section 5 is a more in depth review of the isosurface extraction data structures and algorithms implemented for this paper. Section 6 presents results validating our approach and Section 7 gives some conclusions.

2. Related Work

Isosurfacing is an effective technique used in scientific visualization applications to explore 3D scalar fields. Observations or samples are acquired from a volume by some means and represented digitally on a computer. A scalar field is a discrete sampling in the domain of a continuous function f. The sample values of a scalar field are in the range of f. The locations of the samples form a spatial pattern in the domain and approximations of f can be reconstructed. In this paper we are concerned with regularly sampled scalar fields on rectilinear grids. An isosurface is extracted by passing a reconstruction kernel over the cell set of the scalar field. An isosurface can be defined as follows. Given a discretely sampled scalar field of a function f in $\boldsymbol{R}^3$ find the surface

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG10/151-158



(a) In the span space each cell is represented by a point (min, max). Since $min < v < max$ iff the cell is active, all cells inside the orthogonal dashed lines are active for isovalue v.

(b) The interval representation shows the range of the sampled values $f(x)$, which has a global minimum and maximum. Intervals are represented as a line segment with a min and max values.

Figure 1: Two methods of visualizing cells of a *3D* data set as the span space on the left or a set of intervals on the right.

that satisfies $\{x : f(x) = v\}$, where v is called and isovalue. The extracted isosurface partitions the scalar field into sets, values greater then v and less than v.

A simple and effective isosurfacing method is the *marching cubes* algorithm [LC87] proposed by Lorensen and Cline in 1987. Marching cubes uses a divide and conquer paradigm and is implemented in two phases. The first phase is detection of cells intersected by the isosurface, also known as *active cells*. A linear piecewise approximation of the surface is constructed in each *active cell* in the second phase. The algorithm has a complexity of $O(N)$ since it is necessary to visit each cell in the scalar field in the detection phase. When the input data set is large visiting each cell in the field is too costly. As a result many methods of accelerating this part of the algorithm have been proposed.

In practice, acceleration is achieved by constructing data structures that avoid unnecessary operations on non-active cells, since 30-70% of processing time involves those cells [GW94]. Many data structures and algorithms have been proposed, which fall into three main groups *geometric*, *surface propagation* and *interval bases* approaches. These methods use a variety of strategies to optimize surface reconstruction by avoiding computation of empty cells.

Geometric approaches such as *octrees* [WG92] decompose the cell set into a 3D hierarchy where each node is associated with a *min* and *max* value of the set of cells stored in the node. Surface propagation methods naturally avoids traversal of empty cells as only neighbouring active cells are visited. This method traces the isosurface from a set of seed points and exploit the isosurfaces continuous nature, for example *extrema graphs* [IK94, IK95] or *contour trees* [vKvOB*97].

The final group of algorithms are interval based methods, where each cell in the set is represented as by its min and max values. Some successful early methods were *active lists* [GH90], *span filtering* [Gal91] and *sweeping simplices* [SJ95]. A useful method of viewing intervals is the span space. Many methods based on span space, such as *NOISE* [LSJ96], *ISSUE* [SHLJ96], *interval trees* [CMM*97] and *fixed buckets* [WCJ05], achieved optimal or near optimal search times. Interval based methods are attractive because they offer a compact representation of the data operating in the range of the function rather then the domain. This means interval methods are applicable to a wide variety of data sets with different cell types. For further information a survey [NY06] of marching cubes and derivative work is available.

3. Span Space

The span space [LSJ96] is a novel and compact method for representing the cell set of 3D scalar field. For each cell element there exists an interval $[a,b]$ representing the range of isosurfaces that intersect the cell, where a is the cell's min value and b is the cell's max value. Each cell intersects isosurfaces with a range or *span* of isovalues v, where $a < v < b$. The task of identifying the active cells is reduced to a geometric query of an $\boldsymbol{R}^2$ value space defined by the dimensions *min* and *max*. As illustrated in Figure 1a the cells intersected by isovalue v are contained inside the orthogonal dashed lines. Note that cells can only be mapped to the half space above the line defined by $min = max$ because cell's

maximum values must always be greater then or equal to it's minimum value. An alternative representation of cell intervals in Figure 1b shows v intersecting cells represented as line segments.

In this section we present a number of observations about the span space data representation that lead us to implement the clustering scheme described in Section 4. We base our observations on empirical evidence gathered from a number of data sets from a variety of sources.

3.1. Data Sources

In many application areas much data is heavily quantized. In general, geometric analysis and constructions assume that coordinates are real numbers - i.e. infinite precision floating point numbers. In practice, however, nearly all data sets have finite precision, either in IEEE floating point representation, or by quantization to integers. For 8-bit data (still among the most common), only 256 unique values are possible, and therefore only $\binom{256}{2} = 32640$ different intervals are possible. In a medium-sized data set of 16 million samples, we therefore expect approximately 500 copies of each interval on average, indicating massive redundancy.

In comparison, 12-bit data has approximately 2.1×10^9 possible intervals, so we initially anticipated that savings due to redundancy would be limited to highly quantized 8-bit data. As we shall see this turned out not to be the case.

Our test data consisted of 92 volumetric data sets, sampled at 8 and 12 bits, gathered from a number of sources on the internet. We divided the data sets up into three major categories, experimental (34), medical (34) and simulated (24). The experimental data comes from CT and MRI scans of mostly non-organic objects and materials. These include trees, engines, statues and a variety of other objects. Most of these data sets have large empty cavities, i.e. homogeneous regions in the data. This results in low topological variation in the data and well defined interfaces and transitions between the homogeneous regions. Our medical data is from MRI, CT and PET scans of patients. Medical data has well defined isovalues representing different tissue types and less well defined interfaces due to low gradient transitions between tissue types. Finally, simulated data comes from a range of analytic functions where the exact function is known and we know the data is clean and of course noise is always a problem in real data.

3.2. Redundancy Analysis

We observed in the previous section the potential for massive redundancy in quantized data. Redundancy with respect to quantized data can be defined as the ratio r of the number of unique spans U in the data set to the total number of spans N, $r = \frac{U}{N}$. We can define the percentage of redundancy in the scalar field as follows $R = 100\,(1-r)$. Repeated spans unnecessarily stored in most interval based methods we have investigated increase the total storage cost.

We also observe that zero length spans where $min = max$ are never intersected by any isosurface. The percentage of zero spans Z is computed as $Z = 100(\frac{z}{N})$. Zero spans represent homogeneous regions in the data. An isosurface can never intersected a zero span so the cell can never become active. The interval base methods we have discussed so far unnecessarily store these zero span cells inflating the size of the data structure.

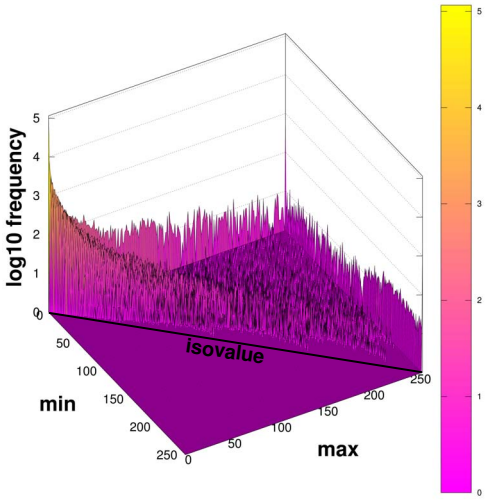
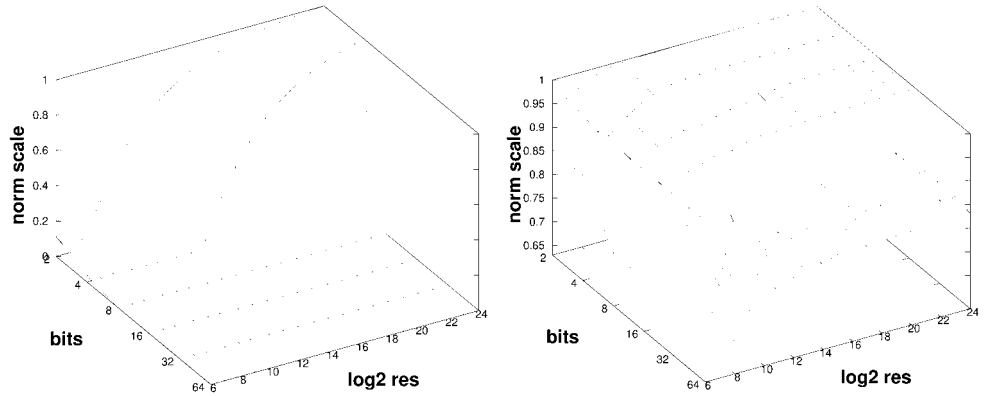


Figure 2: Bivariate histogram of the span intervals for the neghip data set. The vast majority of intervals have small spans, note the similarity between span space and multi-dimensional histograms used to define transfer functions [KD98].

The observations in the span space are best illustrated graphically as a bivariate histogram in Figure 2. We note that a change of variables from (min, max) to $(median, span)$ approximates a plot of isovalue versus gradient as used in transfer function design [KD98]. Due to the limited number of quantized spans and the regular nature of the data, as the size of the cell set increases we expect more cells stack into each bin. We see that the spans are clumped in small value bins and there is massive redundancy in many bins. Sutton et al. [SHwSS00] also used this method to view the span space.

We rapidly discovered immense redundancies due to quantization and to demonstrate these observations further we provide an empirical study of 92 8-bit and 26 12-bit data sets. Table 1 shows a statistical break down of results for 92 8-bit data sets. A high rate of redundancy and large quantity of zero intervals can be seen across the board. However, independent of the source of data, the sheer size of volumetric data sets guarantees a high rate of redundancy and quantization adds to this effect.



(a) Zero Spans (b) Redundancy

Figure 3: For a simple analytic sphere we can see how our measures are affected by increasing resolution and decreasing quantization.

Type	Redundancy	Zeros
All	99%	31.4%
Experimental	99%	44.1%
Medical	99%	25.7%
Simulated	99%	25.2%

Table 1: For 94 data sets quantized to 8-bits we see high redundancy regardless of the data source, this is guaranteed by the size of data sets and quantization.

Having established that 8-bit data is massively redundant, we repeated the experiment with 23 12-bit data sets and compared these to the down-sampled 8-bit versions, expecting that there would be less redundancy and fewer zero span intervals. The results are shown in Table 2. On inspection, most medical data sets actually have well-defined isovalue ranges representing different tissue types, and therefore most cells are broadly uniform in isovalue. This accounts for the high rate of redundancy in real data such as medical and experimental data. However, redundancy in simulated data dropped by 24% in the 12-bit data, compared to 8-bit data. We performed a simple experiment of how these properties change as the number of bits used in the quantization increased.Figure 3 illustrates how our measures are affected by uniformly increasing resolution and increasing number of bits for a simple analytic sphere.

Figure 3a shows that the percentage of zero spans in the data set increases as the resolution increases and decreases as the number of bits increase. This is expected as there are a limited number of spans into which to accumulate, but the number of spans increases as the bit representation increases and the ratio of zero to non-zero spans increases. Figure 3b shows massive redundancy for low bit representation and that redundancy decreases as the number of bits increase in the quantization. As resolution increases, therefore so does redundancy depending on the level of quantization.

	8-bit		12-bit	
Type	Red.	Zeros	Red.	Zeros
All	99%	27.66%	94%	8.98%
Exp.	99%	14.72%	98%	14.72%
Med.	99%	17.86%	95%	5.57%
Sim.	99%	25.14%	75%	4.63%

Table 2: Quantized 8-bit data averaged 98% redundancy, while 12 bit data averaged 94% redundancy: see text.

From these experiments, we reached the following conclusions: there is sufficient redundancy in our sampled data to warrant optimization for data structures based on the span space, even for 12-bit data, and that zero intervals were much more common than we had anticipated. Redundant information in quantized data provides a niche for memory optimization in interval based data structures. However we were concerned about how redundancy would be affected by increasing the number of bits for quantization and how it would effect optimization schemes manipulating redundancy.

4. Volume Clustering & Compression

Given the scalar field of a function f sampled over a volume, the set of N cells that make up the volume is known as the cell set. The number of active cells for a given isosurface v is K. Typically most interval based methods we have investigated sort the entire cell set into their search data structure. This introduces a massive amount of redundant information into the data structure that is unnecessary. Accelerated isosurface extraction can be achieved with a much sparser rep-

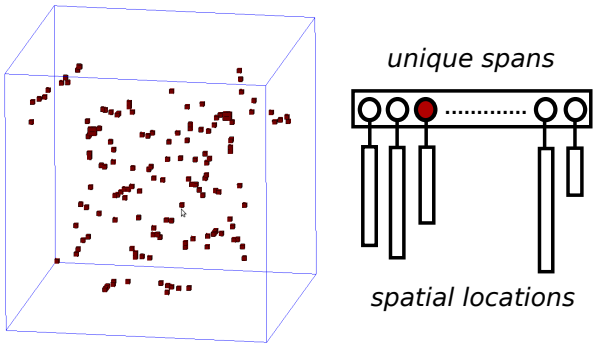


Figure 4: For the neghip data set, on the left, we see 301 occurrences of span $[0,21]$. In our clustered scheme span $[0,21]$ occurs once with a list of references to all the spatial locations in the volume where it occurs. Therefore each reference is only stored once. References may be integer coordinates in the volume or a pointer.

resentation of the cell set created by clustering cells of the same span.

Our optimization scheme exploits redundancy in quantized volume data by clustering all cells sharing a common span and representing them with a single span. We calls this set of unique spans U. Further optimization is achieved by omitting zero-spans, since the corresponding cells are never active in any isosurface. Figure 4 shows our sparse clustered representation of the volume. We chose to represent the clustered cells as list of cells in sorted lexicographic order, meaning spans are stored in ascending order first by the min and then by the max of the span. This is the most convenient way of representing the clusters of cells, for our purposes, as most of the algorithms we tested take as sorted list of cells as input.

An initial pass through our volume data collects unique cells and sorts them in a list. This ordering is maintained throughout the clustering step of the algorithm so a binary search can be used to find spans. If the span has already been collected a simple binary search is done in $O(log(U))$ time to accumulate its location. If the span doesn't exist in the list it is inserted using a quick sort based insertion operation with an average time complexity of $O(Ulog(U))$. These operations must be performed for all N cells in the set meaning the total build time of the clustered cells is $O(Nlog(U)U)$ where $U < N$. The total build time of any interval based method using our clustering scheme is then the sum of the clustering build time and the search data structures build time. In this way we can trade build time for a lower storage cost of $O(U+P)$, where P is the total number of locations of cells.

The only other data compression technique we are aware of is Bordoloi et al. [BS03] who used transform coding and lossy compression to achieve one third the storage of ISSUE and interval tree data structures without impacting isosurface extraction times. However our method is far simpler, easy to implement and plug into any existing interval based approach. Our approach may also be extended to floating point data by quantizing the span space in a similar fashion to the ISSUE [SHLJ96] method.

The drawback of our method is that the natrual topological neighbourhoods of samples implied by the sampling pattern are destroyed by sorting them into our data structure. To rectify this additional information about the six neighbourhood connected components of each sample would need to be stored for each sample location.

5. Interval Data Structure Optimization

To demonstrate our clustering scheme in practice we implemented a number of well established optimal isosurface extraction algorithms based on span space and applied our sparse representation to these data structures. Each algorithms uses a different method to partition the span space to avoid empty cells and achieve optimal search time. We will describe and compare each algorithm and how to apply our clustering scheme. Some of the algorithms described have parallel implementations, ISSUE [SHLJ96] for example, however not all the algorithms describe here are easily implemented in parallel. This means only the sequential versions of the algorithms were tested.

5.1. NOISE

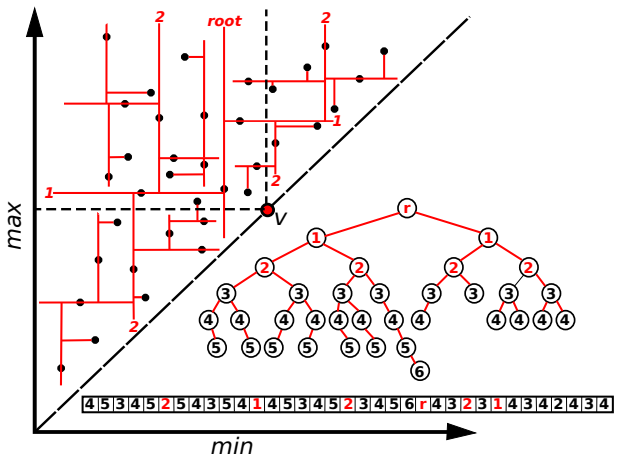


Figure 5: NOISE implements a kd-tree, left, that subdivides the span space for each span. The corresponding binary tree, right, shows the layout of nodes for this tree and the numbers correspond to the levels, nodes marked red are are marked on the span space view. The optimized array view of the kd-tree can also be seen.

The NOISE method, a "near optimal isosurface extraction algorithm", uses a kd-tree [Ben75], a multi-dimensional

binary search tree to partition the cells based on a median discriminator of the *min* or *max* values of the cells in alternating dimensions. A cell that is intersected by the median becomes a node in the tree and all cells less than the median are stored in the left sub-tree and all cells greater than the median are stored in the right sub-tree and the tree is constructed recursively. Each node of the tree represents one cell with a grid reference to it's location in the volume. The memory required by the kd-tree is therefore $O(N)$ as each cell is only stored once with preprocessing time of $O(Nlog(N))$. An optimized search algorithm, that features a pointer-less kd-tree, is provided in [LSJ96] where the kd-tree is built in place by recursively applying quick sorts to subsections of the array representing the cell set. The optimized algorithm yields $O(\sqrt{N}+K)$ search time, where K is the set of active cells. This is the version we have implemented. Figure 5 shows the kd-tree sub division of the span space on the right. To apply our clustering scheme to the NOISE algorithm we simply replace the array of N cells with the clustered array of U unique cells.

5.2. Interval Tree

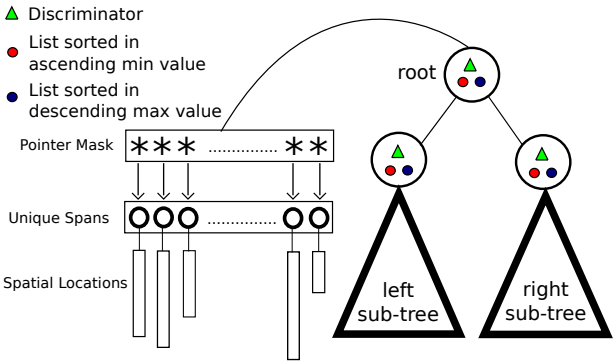


Figure 6: Optimization scheme for the interval tree. The clustered spans are stored externally and an array of pointers to the clusters is passed to the algorithm to avoid duplication of clustered spans inside the data structure.

The interval tree is also a binary search tree that uses a *discriminator* value (often a median value) to divide cells among the nodes of the tree. All intervals spanning the pivot are stored at the root node, see Figure 1b. Intervals wholly less than the pivot are stored in the left child, while intervals wholly more are stored in the right child. Inside each node, two lists are stored, one (left) for searches above the discriminator value, and one (right) for searches below it. To search for active cells at a given isovalue v in a sub-tree, v is compared to the discriminator. If v is less than the discriminator, all intervals in the left list (below the discriminator) are examined and reported if the minimum is below v. The algorithm then recursively searches the left sub-tree. Further optimization is provided by sorting the left list by the minima of the intervals and the right list by the maxima, thereby allowing early termination once all spanning intervals have been determined. This identifies all K active cells in $O(log(N))$ time, with full extraction costing $O(log(N)+K)$, but at the cost of $O(Nlog(N))$ memory and $O(Nlog(N))$ preprocessing time. Moreover, storing each interval multiple times with grid references results in an $O(N)$ term with a large constant, which dominates the memory footprint in practice.

Applying the clustering scheme to the interval tree is only slightly more complicated. Each node in the interval tree represents multiple spans. In Figure 6, we sort a pointers to unique spans in the interval tree to reduce the duplication problem previously that contributes to the interval trees large storage cost.

Comparing the kd-tree and the interval tree we see a number of factors that affect their performance. The interval tree is a much shallower tree than the kd-tree because multiple intervals are stored at each node. A search of the kd-tree must reach a leaf node before the search is terminated because each node represents an interval. The Interval tree however may terminate inside the tree structure without reaching a leaf node if specific criteria are met leading to a shorter search time.

5.3. ISSUE

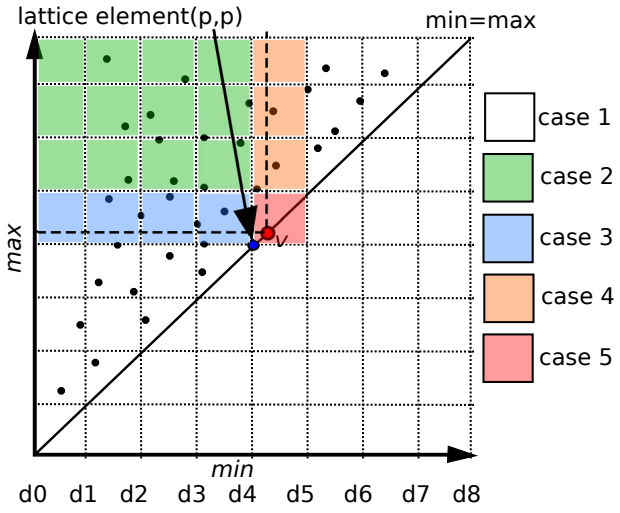


Figure 7: The ISSUE method decomposes the span space into a lattice. This yields a very fast span space search. Using cases 1-5 set of active cells is computed quickly by avoiding extra examination of cells that are definitely intersected by isosurface v.

The ISSUE method, "isosurfacing in span space with utmost efficiency", takes a different approach to dividing up the search space. The algorithm decomposes the span space

into an LxL regular lattice where L is a user specified value. Given L and the global minimum m and maximum M of the data a set of dividing points $\{d_i\}_0^L$ such that d_0, $d_L = \infty$, $d_i < d_{i+1}$ and $\{d_i\}_0^{L-1} \in (m,M]$. Given an isovalue v each lattice element is classified into one of five cases based on its location as in Figure 7. A lattice element is either outside the active area (Case 1), wholly inside the active area (Case 2). The cells that map into Cases 3-5 are potentially active. On the horizontal boundary of the active area, i.e, only the max value of the span must be examined (Case 3), on the vertical boundary of the active area, i.e, only the min value of the span must be examined (Case 4) or on both the horizontal and vertical boundaries (Case 5). There is only one lattice element in Case 5 and both min and max values of the cells must be examined to determine if they are active. This can be done with a simple min-max search, as we have done in this paper, however interval methods can be used to decompose this sub-space further. By restricting the span space search ISSUE will report active cells in $O(K + log(\frac{N}{L}) + \frac{\sqrt{N}}{L})$ time at a cost of $O(N)$ storage and $O(Nlog(N))$ pre-processing time. Clustering can be applied to ISSUE easily by only supplying the clustered cells to the algorithm to sort into the lattice.

ISSUE reports a faster search time complexity then NOISE for the same cost of storage and pre-processing. However, the choice of L has impact on the efficiency of the algorithm. If L is too large or small the algorithm will not perform optimally.

5.4. Fixed Buckets

Isosurface extraction using fixed sized buckets [WCJ05] is an extremely simple and easy to implement algorithm. The input cells are classified by their min and max values in the span space. The cells are then sorted globally by their min values and placed into buckets that contain up to B cells, where B is a user specified value. Note that the last bucket may contain less then B cells. A min-dictionary of the last cell in each bucket is created. Then the cells of each bucket are sorted by their max values. The resulting data structure is a logical 2-D array. Each element is only stored once in the array with total storage cost at $O(N)$ and $O(Nlog(N))$ pre-processing time. The fixed buckets method reports a $O(K+B)$ search time which is comparable to the interval tree. Clustering is easily applied to the fixed buckets method by supplying the set of unique cells as the initial input cells. However the same problem arises as with ISSUE, the choice of B affects the search time.

6. Results & Discussion

Our method of clustering gives a compact representation of the cell set of a volume. We have shown we can collect and organize the unique cells of a volume of $O(Nlog(U)U)$ at a storage cost of $O(U+P)$ which is guaranteed to be less than $O(N)$. This means that we expect the storage cost for clustered versions of NOISE, ISSUE and fixed buckets to be $O(U+P)$ and the interval tree to be $O(Ulog(U)+U+P)$. The saving in storage is achieved at the expense of collecting the unique clusters and constructing the data structure $O(Nlog(U)U+log(U)U)$. Our method is heavily dependant on U so for quantized data we expect to see the clustered versions of the algorithm do better as size of the data sets increase. We also expect our method to do less well for noisy data sets but no worse then the non-clustered versions.

We validate our clustering scheme for each method on 92 8-bit data sets. Figure 8 shows that for an acceptable increase in pre-processing time we see an overall uniform reduction in storage cost when clustering is applied to each data structure. We also see that search time is largely unaffected by clustering. The trend we see is consistent with our initial prediction.

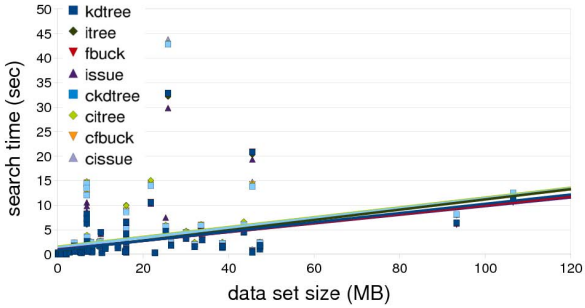


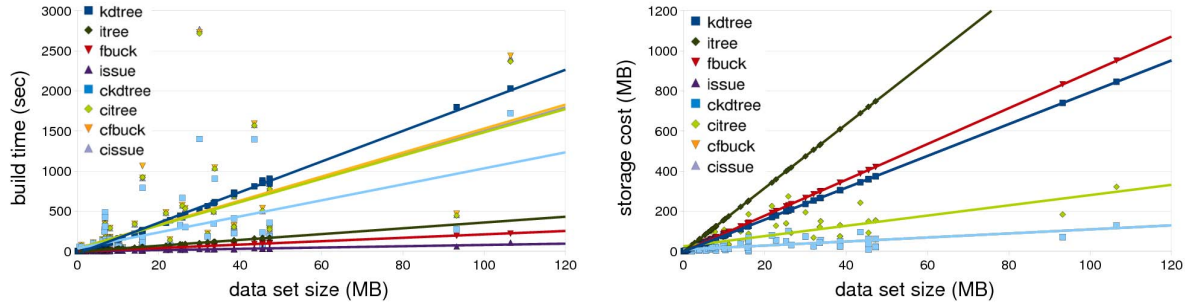
Figure 9: Overall we see a negligible increase in search time due to clustering.

7. Conclusions

We have demonstrated with span space analysis that spans in quantized volume data are massively redundant and many zero spans that are never intersected by any isosurface are present in the data. We have demonstrated that this redundancy depends heavily on quantization and resolution of the data. We have also shown that we can exploit this redundancy to optimize the memory requirements of range based algorithms, for quantized data and proposed methods for dealing with floating points data. The scale of modern data sets guarantees this clustering scheme will work in practice for a variety of data sets. We have also illustrated that knowing the data is important for both design and optimization of data structures and algorithms.

References

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517. 5



(a) Build time for the clustered versions of each algorithm grow at an acceptable rate compared to the non-clustered algorithms.

(b) Storage cost for the clustered versions of the algorithms is greatly reduced compared to their non-clustered counter parts.

Figure 8: Build time and storage cost of the clustered algorithms compared to their non-clustered counter parts. We see and overall reduction in storage cost for an acceptable increase in pre-processing time on the data set.

[BS03] BORDOLOI U. D., SHEN H.-W.: Space efficient fast isosurface extraction for large datasets. *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (2003), 27. 5

[CMM*97] CIGNONI P., MARINO P., MONTANI C., PUPPO E., SCOPIGNO R.: Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 158–170. 2

[Gal91] GALLAGHER R. S.: Span filtering: an optimization scheme for volume visualization of large finite element models. In *VIS '91: Proceedings of the 2nd conference on Visualization '91* (Los Alamitos, CA, USA, 1991), IEEE Computer Society Press, pp. 68–75. 2

[GH90] GILES M., HAIMES R.: Advanced interactive visualization for CFD. *Comput. Syst. Educ. 1*, 1 (1990), 51–62. 2

[GW94] GELDER A. V., WILHELMS J.: Topological considerations in isosurface generation. *ACM Transactions on Graphics 13* (1994), 337–375. 2

[IK94] ITOH T., KOYAMADA K.: Isosurface generation by using extrema graphs. In *VIS '94: Proceedings of the conference on Visualization '94* (Los Alamitos, CA, USA, 1994), IEEE Computer Society Press, pp. 77–83. 2

[IK95] ITOH T., KOYAMADA K.: Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics 1*, 4 (1995), 319–327. 2

[KD98] KINDLMANN G., DURKIN J. W.: Semi-automatic generation of transfer functions for direct volume rendering. *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization* (1998), 79–86. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM, pp. 163–169. 2

[LSJ96] LIVNAT Y., SHEN H.-W., JOHNSON C. R.: A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (1996), 73–84. 2, 6

[NY06] NEWMAN T. S., YI H.: A survey of the marching cubes algorithm. *Computers And Graphics* (2006), 854–879. 2

[SHLJ96] SHEN H.-W., HANSEN C. D., LIVNAT Y., JOHNSON C. R.: Isosurfacing in span space with utmost efficiency (issue). In *IEEE Visualization '96* (1996), Yagel R., Nielson G. M., (Eds.), pp. 287–294. 2, 5

[SHwSS00] SUTTON P. M., HANSEN C. D., WEI SHEN H., SCHIKORE D.: A case study of isosurface extraction algorithm performance. *Data Visualization 2000* (2000), 259–268. 3

[SJ95] SHEN H.-W., JOHNSON C. R.: Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization* (1995), pp. 143–150. 2

[vKvOB*97] VAN KREVELD M., VAN OOSTRUM R., BAJAJ C., PASCUCCI V., SCHIKORE D.: Contour trees and small seed sets for isosurface traversal. *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry* (1997), 212–220. 2

[WCJ05] WATERS K. W., CO C. S., JOY K. I.: Isosurface extraction using fixed-sized buckets. In *Proceedings of EuroVis 2005* (June 2005), Brodlie K., Duke D., Joy K. I., (Eds.). 2, 7

[WG92] WILHELMS J., GELDER A. V.: Octrees for faster isosurface generation. *ACM Trans. Graph. 11*, 3 (1992), 201–227. 2