Computers & Graphics 32 (2008) 704–710

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Space and time efficient isosurface extraction

Slavomir Petrik *, Vaclav Skala

Faculty of Applied Sciences, Department of Computer Science and Engineering, Center of Computer Graphics and Visualization, University of West Bohemia, Univerzitni 8, CZ 306 14 Plzen, Czech Republic

ARTICLE INFO

Article history:
Received 29 February 2008
Received in revised form
17 September 2008
Accepted 21 September 2008

Keywords:
Isosurface extraction
Space efficiency
Data structure

ABSTRACT

The constantly growing size of datasets over the past two decades has put new requirements on the space and time efficiency of isosurface extraction methods. We present a novel approach to fast isosurface extraction, which significantly shortens the preprocessing time and lowers the space requirements. A new computationally inexpensive technique is proposed for transformation of the original volume data into an alternative 1D space. A proper space efficient data structure, built over the transformed data, is used for isosurface extraction. The relative simplicity of the proposed method allows its easy implementation. We demonstrate the low space and time requirements of the proposed approach by a comparison with current state-of-the-art methods applied to real-world datasets.

1. Introduction

Isosurfaces are a standard tool for the investigation of volumetric scalar datasets. The constantly growing size of the datasets in recent decades has put new requirements on the time and space efficiency of isosurface extraction methods.

Although current state-of-the-art isosurface extraction algorithms achieve $O(N)$ space complexity (where N is the total number of cells in a dataset), their run-time space requirements differ significantly. For example, the Interval tree method [1] requires approximately 2 GB of memory for a 512^3 floating-point dataset. Such large space requirements render many of the existing methods for isosurface extraction unusable on workstation-class computers with limited memory resources.

We present a new method for isosurface extraction. The major contributions of this work are the low space and time requirements of the proposed method. Most of the existing techniques use the minimum and maximum values of each cell to identify active cells–cells intersected by the isosurface. We propose a new transformation of a cell's min–max interval into an alternative 1D space. A proper space efficient data structure, constructed over the transformed data, is used during active cell search. Our algorithm outperforms or is comparable with existing methods in all three observed aspects: preprocessing time, data structure size, and performance during the search for active cells.

The transformation of cells' min–max intervals into an alternative 1D space is described in Section 3.1 and isosurface extraction over such 1D space (Section 3.2). Finally, we show comparisons of the method proposed with four other isosurface extraction methods in Section 6. The relative simplicity of the proposed method allows its implementation with reasonably little effort. We believe that the algorithm presented offers a valuable solution to a wider family of range-search applications.

2. Related work

Early geometric-space techniques for isosurface extraction marched through all the cells of a dataset during the search for active cells. A simple method is the Marching Cubes (MC) algorithm, introduced by Lorensen and Cline [2]. A number of solutions have been proposed to resolve the problem of the internal cell ambiguities of the original MC algorithm [3–5]. Wilhelms and van Gelder proposed octree spatial subdivision [6] to accelerate the search for active cells. Surface growing techniques [7] use a set of initial cells, from which all isosurface components can be reconstructed by tracing the isosurface into the neighboring cells.

On the other hand, the value-space methods use only the minimum and maximum value to represent the cells. In the Span Space [8] every cell with minimum value a, and maximum value b, is represented as a point (a, b) on the plane (Fig. 1a). Let us assume that there are N cells in a dataset, K of which are intersected by an isosurface. The NOISE method [8] allows for active cells search in $O(\sqrt{N}+K)$ time, with the aid of the kd-tree subdivision of the Span Space. The ISSUE method [9] uses an $L \times L$ lattice subdivision of the Span Space, and enhances the search time to $O(\log(N/L)+\sqrt{N}/L+K)$.

The Interval tree technique [1] guarantees the worst case optimal run-time efficiency $O(K+\log N)$. The cells are grouped at the nodes of a balanced binary tree according to their minimum–

* Corresponding author. Tel.: +420 377 63 2401; fax: +420 377 63 2402
E-mail address: spetrik@kiv.zcu.cz (S. Petrik).
URL: http://herakles.zcu.cz/people (S. Petrik).

0097-8493/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2008.09.009

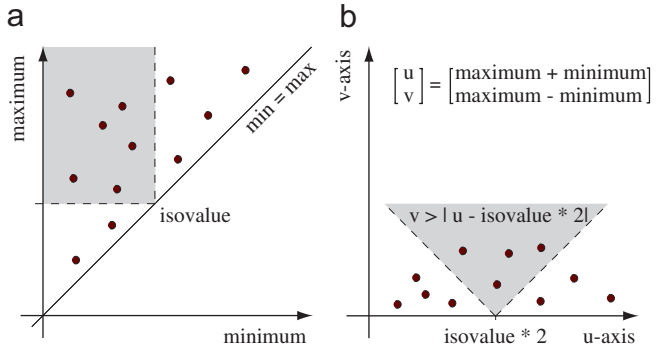

Fig. 1. The Span Space (a) and the *uv*-space (b). Dark-shaded regions contain active cells.

maximum interval and the discriminant value computed for each tree node.

Waters et al. [10] proposed Fixed-sized buckets to organize the list of cells. First, all the cells of the dataset are sorted by their minimum value and divided into buckets of the same size B. Finally, the cells are sorted bucketwise by the maximum value. The active cells can be extracted in $O(K+B)$ time.

This paragraph discusses the space requirements of the value-space methods described above. Let us assume that each of N cells of the dataset is represented by its ID (c bytes), minimum (d bytes), and maximum (d bytes) value. The NOISE and Fixed-sized buckets techniques store the (ID, min, max) record once for each cell. Thus, the run-time space requirements of the NOISE and Fixed-sized buckets are $(c+2d)N$. The ISSUE and Interval tree use two records for each cell: (min, ID) and (max, ID). Therefore, the ISSUE and Interval tree require storage space equal to $(2c+2d)N$. If the cell IDs are represented by 32-bit integers and min–max values by 32-bit floats, then the ISSUE and Interval tree applied to a 512^3 floating-point dataset require approximately 2 GB of memory, while the proposed method lowers this requirement to about one-fourth.

Bordoloi and Shen introduced the space-efficient method [11], which first transforms the cells' extremal values into the 2D *uv*-space (where $u = \text{maximum} + \text{minimum}$ and $v = \text{maximum} - \text{minimum}$, Fig. 1b) and then applies the quantization to divide the *uv*-space into a finite set of $M \times L$ buckets. The method requires the storage space for N cell IDs and $ML+L+1$ quantization levels, where $ML+L+1$ is typically equal to $N/100$. Depending on the level of quantization, a certain amount of false positives can occur during the search for active cells.

Another possible enhancement of space efficiency can be achieved through dimensionality reduction methods such as principal component analysis (PCA) [12]. However, transformation of the maxima and minima values using PCA faces the problems of a highly inaccurate search for cells and long preprocessing times.

The method described in this paper is based on a simple transformation of the {minimum, maximum} information of each cell into a 1D space. The proposed transformation lowers the worst-case space requirement to $2N$ words. Construction of the search structure over 1D data requires only one initial $O(N \log N)$ sorting step, which shortens and simplifies the preprocessing compared to the existing methods. Our algorithm achieves the near-optimal search times, while the number of false positives during the active cell search decreases significantly compared to [11].

Once identified, the active cell IDs are transferred into our point-based visualization system. A range of point-based techniques has been developed for isosurface rendering [13–19]. We have adopted a technique similar to iso-splatting [15] for interactive focus+context [20] visualization accelerated by modern GPU hardware. For high-quality isosurface rendering we use the method of Brentzen and Christensen [13].

3. Search structure

Many existing isosurface extraction algorithms build a data structure within which the minimum and maximum values of cells are stored separately. However, the storage space required by those algorithms may grow up to four times the size of the original dataset. Thus, the first step of the proposed method is to transform the {minimum, maximum} pairs of cells into the space-efficient 1D form (Section 3.1). At run-time, the search for active cells is performed in the 1D space, with the aid of the simple data structure described. The search algorithm is explained in Section 3.2.

3.1. Transformation

The main idea of the transformation is to convert the {minimum, maximum} pair of each cell into the parameter t (denoted t_c for cell c):

$$\{c_{min}, c_{max}\} \rightarrow t_c, \quad t_c \in [0,1] \tag{1}$$

The transformation (Eq. (1)) is done by quantization of the max-axis of the Span Space [8] to the M quantization intervals. M is the parameter to our method and its choice is explained later in Section 4. After the max-axis quantization the Span Space becomes a finite set of parallel horizontal lines. Parameter t from the transformation, is equal to 0 at the beginning of the bottom-most line, and is equal to 1 at the right end of the top-most line. The t-interval per one max-axis quantization interval is $t_{int} = 1/M$.

Considering the quantization described in the previous paragraph, the parameter t_c of a cell c is computed as follows:

$$t_c = t_i + t_0 \tag{2}$$

where

$$t_i = \left\lfloor \frac{c_{max} - \max_{min}}{\max_{max} - \max_{min}} \Big/ t_{int} \right\rfloor * t_{int} \tag{3}$$

$$t_0 = \frac{c_{min} - \min_{min}}{\min_{max} - \min_{min}} * t_{int} \tag{4}$$

Such a 1D index is used for a fast construction of the search structure and for the identification of active cells. As can be seen, the transformation does not handle both extremal values in a symmetric way (only the max values are quantized), which results in the small search error rates of the method presented, see Section 6.2.

Once the parameter t is computed for each cell of a dataset, the cells are sorted by t in increasing order. Finally, a list of records is constructed. Each record contains parameter t and a list of IDs of the cells, which have this value of parameter t.

Since the list of cells is sorted by the t parameter, the records can be created by simply traversing the list of cells and grouping the cells with the same parameter t into the same record. An index of the first record on each max-axis quantization interval is placed into a simple search dictionary which helps during the active cell search. Note, that Eqs. (2)–(4) can be implemented very efficiently using bit shifts if the M, $\max_{max} - \max_{min}$ and $\min_{max} - \min_{min}$ are powers of 2.

3.2. Extraction

The goal of the extraction phase is to identify all active cells. For a supplied isovalue q, a cell c is defined to be *active* if: $c_{min} \leqslant q \leqslant c_{max}$.

Active cells for the supplied isovalue are collected by traversing the max-axis quantization intervals in the top–bottom order using two nested loops. The outer loop traverses the items of the search dictionary to determine the index of the first record on the current quantization interval. The inner loop collects the active cells from the records in the current quantization interval, until the

condition $t_c \leqslant t_{limit}$ holds. The value of t_{limit} for the n-th quantization interval is computed as follows:

$$t_{limit} = (n * t_{int}) + \frac{q - \min_{min}}{\min_{max} - \min_{min}} * t_{int} \quad (5)$$

The index of the last traversed quantization interval, "*final*", is determined by the selected isovalue q:

$$final = \left\lfloor \frac{q - \max_{min}}{\max_{max} - \max_{min}} \Big/ t_{int} \right\rfloor \quad (6)$$

In the *final* quantization interval we are unable to decide whether the original maximum value of a cell is above or below isovalue q (i.e. whether a cell is active or not), which may introduce a small positive search error.

For small changes of isovalue q, it is often efficient to extract the active cells *incrementally*. For this purpose, we keep list L of the last visited record on each quantization interval. The intervals between the topmost one and the $final_{new}$ are marched from the record with index stored in L, activating the cells until the first inactive record is met. Cells from all the quantization intervals between $final_{old}$ and $final_{new}$ are deactivated.

Similarly, when the isovalue is decreased to q_{new}, the quantization intervals are traversed backwards from the position stored in L, deactivating the cells passed along the way to the new value of $limit_n$. Additionally, full extraction has to be done for the quantization intervals $final_{old}$ to $final_{new}$.

4. Number of quantization intervals

As stated in Section 3.1, the number of max-axis quantization intervals is an optional parameter to the method presented and depends on the data type of the processed dataset.

For datasets with byte data, it is sufficient to quantize the max-axis of the Span Space into 256 intervals to recognize clearly each possible maximum value. In other words, the $\mathscr{R}^2 \to \mathscr{R}$ transformation (Eq. (1)) does not quantize the maximum values of cells (i.e. it preserves the original {minimum, maximum} information), and the active cells can be extracted with zero search error. The same is the situation for 2-byte integer datasets and the search structure with 65 536 quantization intervals.

The number of quantization intervals for the floating-point data has been determined experimentally. The relationship between the chosen number of quantization intervals and the search error has been observed. In all performed measurements, the search error drops under 0.3% for more than 2^{16} quantization intervals, regardless of the input dataset (Fig. 5). Thus, for floating-point data we use 2^{16} quantization intervals. Comparison of the search error with the method [11] is provided in Section 6.2.

5. Optimization

For floating-point datasets the number of different t parameters of the transformed cells can be very large, which may results in a large number of records. In the worst case, there is one record created for each cell. Therefore, we employ user-selected level of the t quantization Δt. Mathematically, the cells of any record $R = (c_0, \ldots, c_n)$ satisfy conditions: $t_{c_0} \leqslant t_{c_1} \leqslant \cdots \leqslant t_{c_n}$ and $|t_{c_n} - t_{c_0}| < \Delta t$. All cells of R are further represented by the parameter $t_R = t_{c_0}$.

Let us assume two records R_1 and R_2 on the same quantization interval, so that $t_{R_1} < t_{limit} < t_{R_2}$. Because each cell of R_2 has its $t > t_{R_2}$, R_2 cannot contain any active cells with $t < t_{limit}$. Therefore, collecting the active cells from records with $t < t_{limit}$ guarantees that all active cells will be found when quantization is applied, avoiding cracks in the isosurface.

6. Comparisons and results

We have implemented the ISSUE [9], Interval tree [1], Fixed-sized buckets [10], and Quantized search [11] algorithms to compare the method proposed against existing methods. The Span-triangle method [21] was omitted from the tests because it is aimed only to quantized data (byte or 16-bit integer). First, the formal space and time complexities of the presented method are discussed, followed by the results of the tests.

6.1. Formal complexity analysis

The construction of our data structure involves three steps. The transformation of {minimum, maximum} values into t parameter requires $O(N)$ time, while the sorting pass requires $O(N \log N)$ time. The creation of records is $O(N)$ step. Thus, construction of our search structure can be done in $O(N \log N)$ time.

The most time-consuming parts of the construction phase are the sorting steps. The ISSUE method [9] requires two sorting passes for each non-empty lattice element (row and column data structures). The Interval tree method [1] needs to sort AL and DR lists of cells within each tree node. The Fixed-sized buckets method [10] sorts all cells by minimum value in $O(N \log N)$ time, and then resorts the cells bucketwise by maximum value in $O(B \log B)$ time for each of the $N/B + 1$ buckets (where B is the number of cells per bucket). In the Quantized search method [11] all cells are first sorted by their u value, then the u axis is quantized into M intervals and cells are sorted for the second time within each u-interval.

Since there is only one initial sorting step in the proposed method, the construction time is significantly shorter when compared to the current state-of-the-art algorithms for active cell identification. This conclusion is supported by the measurements presented in Section 6.2.

At run-time, active cells are identified by traversing the list of records. In the worst-case scenario, there is one record created for each cell. In such a case, we need to visit $K + E$ records, to extract K active cells. For each record visited, one comparison of its t_R parameter is performed. For each traversed quantization interval one check of the search dictionary and one failed t_R comparison are performed. E is the search error introduced by the small amount of records visited in the last quantization interval, which do not contain active cells. Because E is very small in practice (under 0.3%), the run-time complexity of our search algorithm is $O(K)$.

In a typical case, many records contain more than one cell. Therefore, to extract K active cells we usually need far less than K comparisons, which shortens the search process.

Finally, the space requirements of the proposed method are discussed. For comparison purposes, Table 1 summarizes the final

Table 1
The space requirements of the tested methods

Method	Space requirement (words)	Space complexity
ISSUE	$4N$ + kd-trees at min = max	$O(N)$
Interval tree	$4N$ + tree overhead	$O(H + I)$
Fixed-sized buckets	$3N$ + min dictionary	$O(N)$
Quantized search	$N + ML + M + 1$	$O(N)$
Our method	$2N$[a]+search dictionary	$O(N)$

N = # of cells, H = # of interval tree nodes, I = # of distinct intervals in data, M, L = # of quantization levels.

[a] The worst case.

space requirements of all the tested methods in the number of *words*. The Fixed-sized buckets method [10] stores the (ID, min, max) = (c bytes, d bytes, d bytes) record once for each cell. Assuming that $c = 4$ bytes (one word) and $d = 4$ bytes (one word), we can say that the space requirements of [10] are $3N$ words. As shown in Table 1, the reported space complexity of the tested methods is usually $O(N)$; however, their run-time space requirement ranges from $2N$ to $4N$.

Because our method stores the ID number and t parameter once for each cell, the temporal space required by our method is $2N$ words. However, due to the bucketization of cells into records, the final space requirements of the proposed method are between $1N$ and $2N$, which is competitive with the method of Bordoloi and Shen [11] (see results in Section 6.2).

There is also a small space required for the search dictionary, which contains one 4-byte integer for each quantization interval. Thus, the space allocated for the search dictionary is $256 * 4$-bytes $= 1$ kB for byte data, and 256 kB for 16-bit integer and floating-point datasets.

Table 2
Summary of datasets used during tests

Dataset	Resolution	Description
Skull	$256 \times 256 \times 256$	Hexahedral grid, byte
CT-head	$256 \times 256 \times 113$	Hexahedral grid, 16-bit integer
FiveJets	$128 \times 128 \times 128$	Hexahedral grid, 32-bit float
TeraShake	$750 \times 375 \times 100$	Hexahedral grid, 32-bit float
Isabel	$500 \times 500 \times 100$	Hexahedral grid, 32-bit float
Vertebra	$512 \times 512 \times 512$	Hexahedral grid, 16-bit integer
X^2-Y^2	$512 \times 512 \times 512$	Hexahedral grid, 32-bit float

6.2. Tests

The analyses from Section 6.1 are supported by the results of measurements. Table 2 summarizes seven datasets used during tests. The tests for Vertebra and X^2-Y^2 datasets were done on the 64-bit Intel processor and 4 GB of RAM. The tests for all other datasets were done on a desktop PC with Intel 3.2 GHz processor, 2 GB of RAM and ATI FireGL V5200 graphics adapter.

Fig. 2 provides a comparison of the construction time of the presented data structure versus four tested methods. For testing purposes, the number of quantization intervals for our method

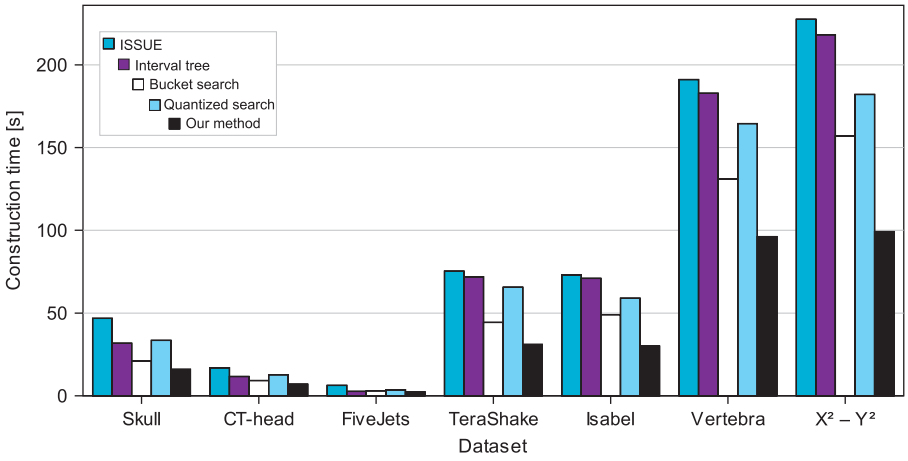

Fig. 2. Comparison of the construction times. As predicted by the analysis of construction complexity, the search structure proposed achieves the shortest construction times of all tested methods.

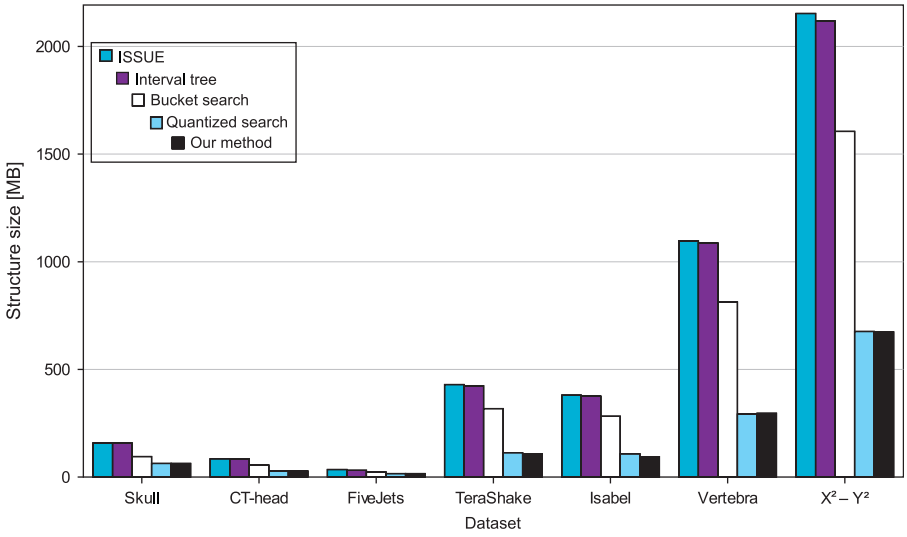

Fig. 3. Comparison of the size of the final data structures. Only the Quantized search achieves size of the data structure comparable to our method, but at the cost of almost twice as long construction time (Fig. 2) and higher search error.

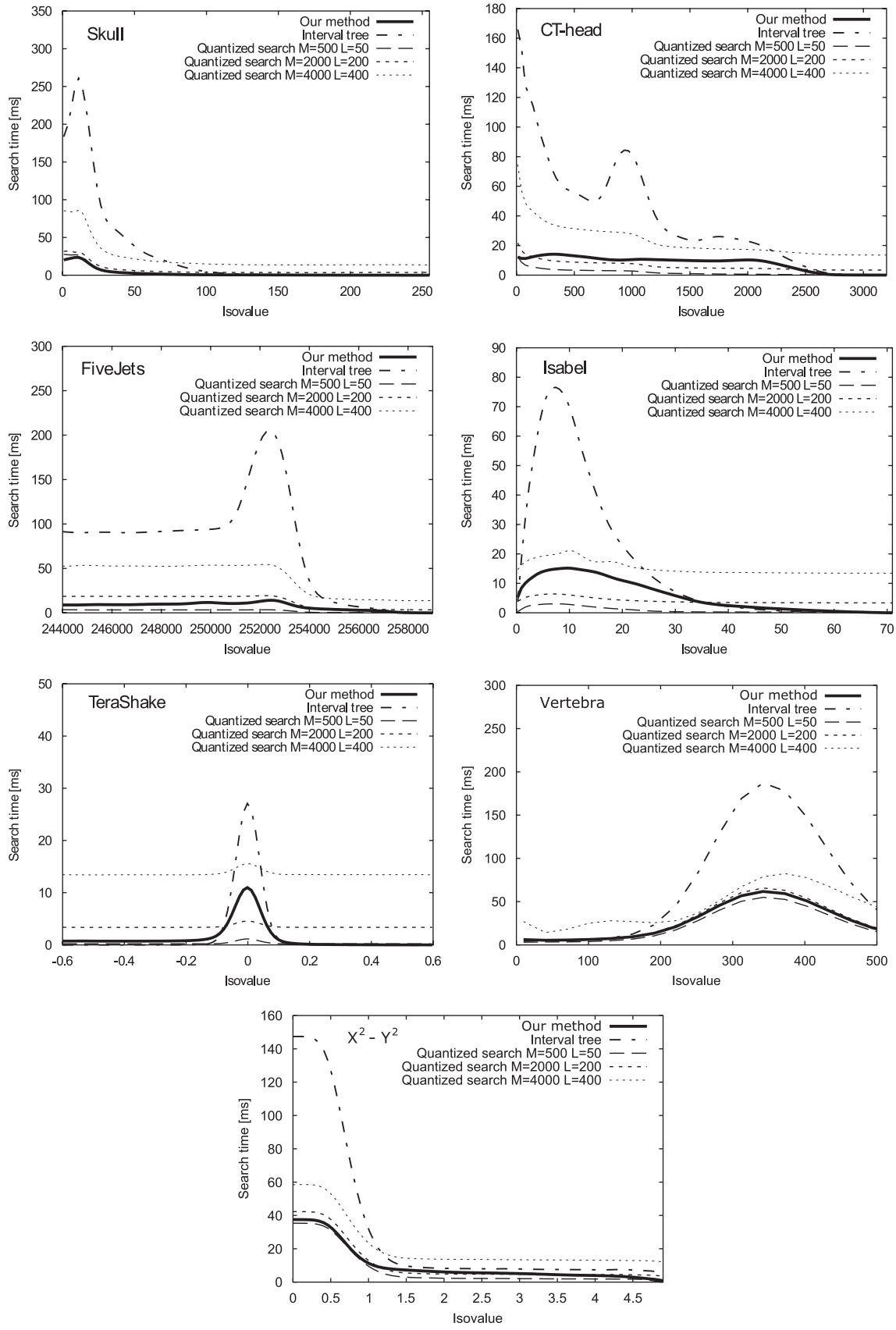

Fig. 4. Comparison of the search times. The plots show that the search times of our method are comparable with Quantized search with $M = 2000$ and $L = 200$ quantization levels. As mentioned in Section 6.1 the run-time complexity of our proposed method is $O(K)$.

has been determined according to the data type of a dataset (Section 4). For the Fixed-sized buckets [10] the fixed bucket size of 8192 cells has been used as recommended by the authors of the method. For the Quantized search the $(M,L)=(2000,200)$ quantization levels were constructed, because at this setup it achieves the run-time performance similar to our method (see Fig. 4). As predicted by the analysis of the construction complexity, our search structure achieves the shortest construction times of all tested methods. Construction times provided by Fig. 2 include creation of the temporary and final data structure without data loading.

Fig. 3 compares the size of the final data structures. Because we do not compress the cell IDs, the size of our search structure is always greater than $1N$ words. We need additional space to store the t_R parameter of each record, and the space for the search dictionary. Thus, the total space required by our method is between $1N$ and $2N$ words for the tested datasets. Only the Quantized search [11] achieves a comparable size of data structure. However, our method achieves much better construction times than the Quantized search and lower search error.

Next, the measurements of the search times are provided. The search times of the presented method are compared against the Interval tree and the Quantized search with three different setups of quantization levels. The Interval tree method has been chosen because of its reported near-optimal search time $O(K+\log N)$, where K is the total number of active cells. The Quantized search has been included into the measurements because, as well as the method presented, it uses quantization of the data to shorten the search time. For each dataset, 1000 isovalues from the value range of the data were randomly chosen. For each chosen isovalue, the average search time for 100 full extraction queries was recorded.

Fig. 4 shows that our method outperforms the Interval tree. In fact, the search times of our method are comparable to those of a Quantized search with $M=2000$ and $L=200$ quantization levels. However, as will be shown later in this section, the search error of the Quantized search for $(M,L)=(2000,\ 200)$ is significantly higher when compared with our method.

Note that the authors of the Quantized search method [11] do not provide any specific recommendation for the optimal setup of M and L parameters, while the value of the optional parameter in our method (the number of max-axis quantization intervals) is exactly given by the data type of the processed dataset (see Section 4).

In the following paragraphs, the search error of our method is discussed. The search error of the method presented is introduced by the quantization of the original maximum values of the cells during the transformation (Section 3.1). Therefore, on the last traversed quantization interval *final*, we are unable to decide whether the original maximum value of a cell was above or below the user-specified isovalue. Fig. 5 shows that the search error decreases with an increasing number of quantization intervals. We used three floating-point datasets with a different value range. For all three datasets, the search structures with 2^{10}–2^{19} quantization intervals were constructed. For each constructed data structure, the average search error for 200 full extraction queries (with random isovalues from the value range of the data) was recorded. The results indicate that the average search error remains below 0.3% for a search structure with more than 2^{16} quantization intervals, regardless of the input dataset.

Finally, Table 3 provides exact measurements of search error for floating-point datasets and selected isovalues. Note that the search error of our method is zero for byte and 16-bit datasets, because each possible maximum value is covered by one quantization interval; thus, there is no quantization of the input data at all. The search error of the Quantized search method has been measured for various quantization levels ranging from $(M,L)=(500,50)$ to $(4000,\ 400)$. As can be seen, our method achieves much lower search error, even compared to the Quantized search with $M=4000$ and $L=400$ (Fig. 6).

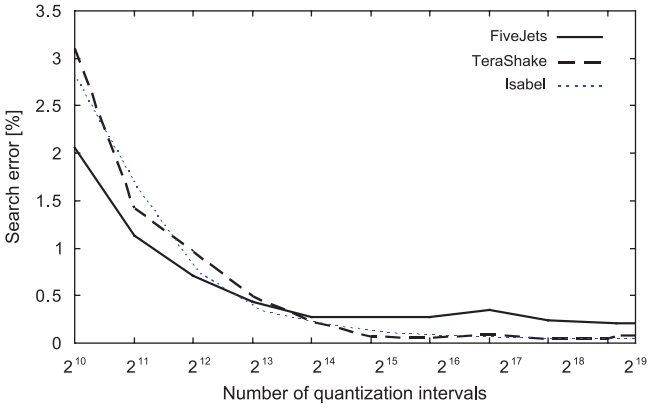

Fig. 5. Plot of the search error versus number of quantization intervals for floating-point datasets. The results show that the search error falls below 0.3% for search structure with more than 2^{16} quantization intervals, regardless of the input dataset.

Table 3
Comparison of the search error for the Quantized search and our method

Dataset/isovalue	Quantized search (%)			Our method (%)
	$M=500$, $L=50$,	$M=2000$, $L=200$,	$M=4000$, $L=400$	
FiveJets/253947.5	1.81	0.86	0.97	0.025
TeraShake/0.015	6.25	3.05	1.25	0.04
TeraShake/0.08	5.29	4.95	4.26	0.05
Isabel/16.3	1.63	0.53	0.48	0.18
Isabel/25.6	5.44	1.57	1.25	0.21
$X^2-Y^2/3.5$	3.61	2.77	1.01	0.17
$X^2-Y^2/4.1$	2.81	2.01	1.32	0.31

7. Conclusions and future work

We have presented a novel space efficient method for isosurface extraction. The core of the method is a novel transformation of the original volume data into an alternative 1D space. A simple data structure, built over transformed 1D data, is used to identify active cells during isosurface extraction.

We have shown that the data structure presented achieves a lower size than the space efficient method created by Bordoloi and Shen [11]. Moreover, the method presented requires shorter construction time than the existing state-of-the-art techniques, while achieving near-optimal search times and smaller search errors.

The space and time efficiency of the method presented is a valuable benefit for isosurface extraction especially today, when the size of the datasets is growing steeply. Moreover, the generality of our method offers an attractive solution for a wider range of interval-search problems.

There are many possible directions for future work. We are developing an extension of the method presented for time-varying datasets.

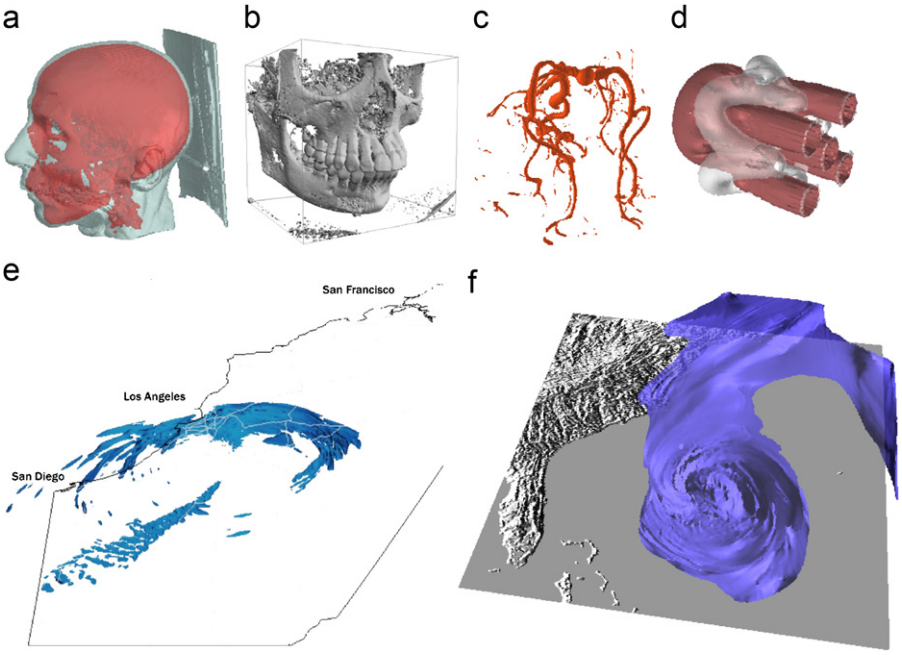

Fig. 6. Isosurfaces extracted using the method presented. (a) CT-head dataset, red isovalue = 1879, transparent isovalue = 850. (b) Skull dataset, isovalue = 51. (c) Vertebra, isovalue = 1000. (d) FiveJets dataset, red isovalue = 254 052, transparent isovalue = 251 200. (e) TeraShake dataset (seismic wave velocity × isovalue = 0.052, timestep 80). (f) Isabel dataset (wind velocity magnitude isovalue = 22, timestep 4).

Acknowledgments

This work has been supported by the project VIRTUAL no. 2C 06002 Ministry of Education of the Czech Republic. Skull and Vertebra datasets were downloaded from www.volvis.org. FiveJets dataset is from Time-Varying Volume Data Repository of UC Davis. CT-head and MR-brain datasets are from Stanford Volume Data Archive. TeraShake and Isabel datasets were provided by the San Diego Supercomputer Center. X^2-Y^2 dataset has been generated at the Center of Computer Graphics and Visualization at University of West Bohemia, Czech Republic. Detailed information can be found at http://herakles.zcu.cz/projects/. Authors would also like to thank their colleagues for consultations and comments.

References

[1] Cignoni P, Marino P, Montani C, Puppo E, Scopigno R. Speeding Up Isosurface Extraction Using Interval Trees. IEEE Trans. Visualization and Computer Graphics 1997;3(2):158–70.
[2] Lorensen WE, Cline HE. Marching Cubes: a high resolution 3D surface construction algorithm. In: Proceedings of the SIGGRAPH, 1987. p. 163–69.
[3] Cignoni P, Ganovelli F, Montani C, Scopigno R. Reconstruction of topologically correct and adaptive trilinear isosurfaces. Computers & Graphics 2000;24(3):399–418.
[4] Montani C, Scateni R, Scopigno R. A modified look-up table for implicit disambiguation of Marching Cubes. The Visual Computer 1994;10(6):353–5.
[5] Schaefer S, Warren J. Dual Marching Cubes: primal contouring of dual grids. In: Proceedings of the Pacific graphics, 2004. p. 70–6.
[6] Wilhelms J, van Gelder A. Octrees for faster isosurface generation. ACM Transactions on Graphics 1992;11(3):201–27.
[7] van Kreveld MJ, van Oostrum R, Bajaj CL, Pascucci V, Schikore D. Contour trees and small seed sets for isosurface traversal. In: Proceedings of the symposium on computational geometry, 1997. p. 212–20.
[8] Livnat Y, Shen HW, Johnson CR. A near optimal isosurface extraction algorithm using the span space. IEEE Transactions on Visualization and Computer Graphics 1996;2(1):73–84.
[9] Shen HW, Hansen CD, Livnat Y, Johnson CR. Isosurfacing in span space with utmost efficiency (ISSUE). In: Proceedings of the IEEE visualization, 1996. p. 287–94.
[10] Waters KW, Co CS, Joy KI. Isosurface extraction using fixed-sized buckets. In: Proceedings of the EuroVis, 2005. p. 207–14.
[11] Bordoloi U, Shen HW. Space efficient fast isosurface extraction for large datasets. In: Proceedings of the IEEE visualization, 2003. p. 201–8.
[12] Jolliffe IT. Principal component analysis. New York: Springer; 2002.
[13] Brentzen JA, Christensen N. Hardware accelerated point rendering of isosurfaces. In: Proceedings of the WSCG, 2003.
[14] Cline HE, Lorensen WE, Ludke S, Crawford CR, Teeter BC. Two algorithms for the three-dimensional reconstruction of tomograms. Medical Physics 1988;15(3):320–7.
[15] Co CS, Hamann B, Joy KI. Iso-splatting: a point-based alternative to isosurface visualization. In: Proceedings of the Pacific graphics, 2003. p. 325–34.
[16] Grossman JP, Dally WJ. Point sample rendering. In: Proceedings of the rendering techniques (Eurographics), 1998. p. 181–92.
[17] Levoy M, Whitted T. The use of points as display primitives. Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill; 1985.
[18] Livnat Y, Tricoche X. Interactive point-based isosurface extraction. In: Proceedings of the IEEE visualization, 2004. p. 457–64.
[19] Rusinkiewicz S, Levoy M. QSplat: a multiresolution point rendering system for large meshes. In: Proceedings of the SIGGRAPH, 2000. p. 343–52.
[20] Wang L, Zhao Y, Mueller K, Kaufman A. The magic volume lens: an interactive focus+context technique for volume rendering. In: Proceedings of the IEEE visualization, 2005. p. 367–74.
[21] von Rymon-Lipinski B, Hanssen N, Jansen T, Ritter L, Erwin K. Efficient point-based isosurface exploration using the span-triangle. In: Proceedings of the IEEE visualization, 2004. p. 441–8.