Optimal hybrid memory constrained isosurface extraction

Jürgen Toelke, Dietmar Saupe
Department of Computer and Information Science
University Konstanz, Germany

March 2006

Abstract

Efficient isosurface extraction from large volume data sets requires special algorithms and data structures that allow to quickly identify large parts of the data set, that do not contain any part of the surface and which can be eliminated from the search. Such algorithms typically use a hierarchical spatial subdivision of the volume or they organize the scalar values attached to the cells of the volume, i.e., intervals, in some suitable data structures. Octrees, kd-trees, and interval trees are commonly applied. However, these auxiliary data structures demand storage space that can be several times as large as the original volume data itself. In practise, memory capacity is constrained, preventing the application of the most efficient data structures for extraction of isosurfaces from large volume data sets. For such cases out-of-core methods provide a solution, however, at the cost of disk block reading operations. We present a hybrid algorithm that constructs an optimal data structure within the memory constraint by combining binary space partition (bsp) trees with fast search methods at some leaf nodes of the bsp-tree and memory-free linear search or out-of-core methods at the remaining leaf nodes. The method optimally trades off space for extraction speed. We develop the theory for the optimization, provide implementation details and examples demonstrating the efficiency of the approach. To perform the optimization, we develop and apply models for calculating the memory and estimating the expected extraction time for the search methods based on auxiliary data structures and for an out-of-core method.

1 Introduction

1.1 Isosurface extraction for volume data on rectilinear grids

For visualization of scalar volumetric data, isosurfaces provide a basic intuitive approach. The process of generating an isosurface proceeds in several steps. In the first step the volume data is scanned and all those volume cells that contain a part of the isosurface are reported. Secondly, a polygonal surface representation for each of these cells is generated. Finally, the polygons are rendered and displayed. In interactive applications, the isovalue defining the surface may be changed by the user and new isosurfaces need to be computed on the fly. Due to the large amount of searching and computing, an interactive investigation of volume data with isosurfaces is difficult even for volume data of moderate size. Therefore, complexity reduction of isosurface computation has been an important issue of research in recent years.

Formally, an isosurface for a scalar valued function $f : D \to \mathbb{R}$ with $D \subset \mathbb{R}^3$ is given by the preimage $f^{-1}(t)$ for some isovalue $t \in \mathbb{R}$. Strictly speaking, there is no guarantee that the preimage really is a surface without posing conditions on the function f. For example, $f^{-1}(t)$ is manifold, if $f : D \to \mathbb{R}$ is continuously differentiable on an open set D and t is a regular value of f. However, in practice, f is given only in terms of a finite collection of point samples, and this issue is generally ignored. The structure of volume data may be regular or irregular. In the regular case, we have samples of function values on a rectilinear grid so that the data can be regarded as a three-dimensional array of scalar values. For example, medi-

cal 3D imaging modalities typically produce such regular volume data. Irregular volume data often arise from finite element simulations. In this paper, we focus on the predominant case of regular rectilinear volume data.

For simplicity and without loss of generality, let us label the grid points using integer coordinates (i, j, k). Grid points are also called *voxels*. The samples $v_{i,j,k} := f(i, j, k) \in \mathbb{R}$ at the grid points are called *voxel values* and a *cell* in the 3d-grid has eight voxels as its corner grid points. For a cell $C_{i,j,k}$ with voxel (i, j, k) at its lower left front corner, the corresponding eight voxel values are collected in a set $V_{i,j,k}$. A continuous interpolation $\hat{f}$ of these voxel values to the points in the cell can be taken as a model for the underlying volume function f within the cell. This interpolation is required to have the property, that the values of $\hat{f}$ in a cell are restricted to the interval spanned by the corresponding voxel values. Together with the continuity of $\hat{f}$, we therefore have for the range of $\hat{f}$ in a cell $C_{i,j,k}$

$$\hat{f}(C_{i,j,k}) := \{f(x,y,z) \mid (x,y,z) \in C_{i,j,k}\} = [v_{\min}, v_{\max}],$$

where

$$v_{\min} = \min V_{i,j,k}, \quad v_{\max} = \max V_{i,j,k}.$$

Now let an isovalue $t \in \mathbb{R}$ be given. Then the first step of the isosurface computation, namely the cell extraction, amounts to reporting all cells with t contained in the corresponding interval $[v_{\min}, v_{\max})$. These cells are called active cells. All other cells can be regarded as "empty", containing no part of the isosurface.

Indeed, it suffices to consider the half-open intervals $[v_{\min}, v_{\max})$ in place of the closed interval $[v_{\min}, v_{\max}]$, which is used by some authors. With half-open intervals we have the following advantages. It forces an extraction algorithm to report only one cell in place of two in cases where the isosurface only touches a face of a cell. Also there are theoretical reasons to favor using the half-open interval. Without further assumptions, we cannot conclude that $\hat{f}^{-1}(t)$ is a surface. To solve this problem, we may introduce a regularization. It can be shown that $\hat{f}^{-1}(t+\varepsilon)$ is a polygonal surface for all sufficiently small $\varepsilon > 0$, when a suitable class of interpolation functions is used. Such interpolation functions arise for the approach of marching tetrahedra[16] for isosurface construction. We cannot give more details here and refer to the work on simplicial algorithms[2].

In the next subsection, we review the state-of-the-art for isosurface extraction, and in the following one, we address the memory constraint for the extraction process and introduce our approach for designing hybrid methods that optimally adapt to the limited memory.

1.2 State of the art

Bajaj, Pascucci, and Schikore[3] provided a survey over different methods to organize the extraction as efficiently as possible. Another overview is given by the same authors[5], including a proof, that $O(k + log(n))$ is the lowest possible worst-case extraction time for any method based on isovalue comparisons, where n is the size of the volume data and k is the size of the output. We continue by reviewing the major isosurface extraction methods.

The canonical and elementary method for the cell extraction problem proceeds by *enumeration*, i.e., it sequentially visits all cells $C_{i,j,k}$ in the volume and checks the condition $t \in [v_{\min}, v_{\max})$. This method was used in the "marching cubes" paper of Lorenson and Cline[15]. Besides storage for the original volume data, this method does not require any additional memory except a small constant amount for a few variables, while the cell extraction time is linear in the size of the volume data, yielding a linear time complexity of $O(n)$.

With a *hierarchical spatial subdivision* of the volume, a considerable acceleration can be achieved at the cost of moderate additional CPU storage space. Wilhelms and Van Gelder[22] proposed an *octree subdivision* for this purpose. Each node n in the tree corresponds to a rectilinear block B of cells $C_{i,j,k}$ in the volume data and is attributed by an interval $[n_{\min}, n_{\max})$, which is the union of all intervals $[v_{\min}, v_{\max})$ of cells $C_{i,j,k}$ from the block B belonging to node n. In other words, $[n_{\min}, n_{\max})$ denotes the range of values the interpolation function $\hat{f}$ attains in the portion of the volume corresponding to the node n. The cell extraction is organized along a recursive tree traversal in which all those branches can be ignored, which are rooted at a node n with $t \notin [n_{\min}, n_{\max})$. Leaf nodes n with $t \in [n_{\min}, n_{\max})$ are investigated cell by cell as in the enumeration algorithm.

More substantial acceleration of the cell extraction phase can be provided by methods that rely on partitioning the volume into subsets each of which consists of a (generally disconnected) union of cells $C_{i,j,k}$ with simi-

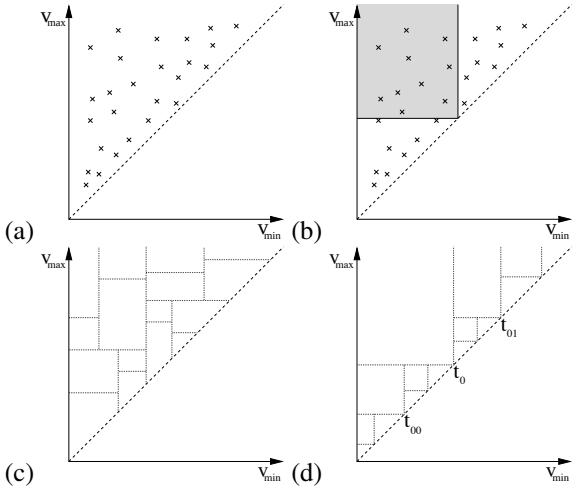


Figure 1: (a) top left: intervals as points in span space. (b) top right: query with isovalue t in span space. (c) bottom left: The partitionings of span space according to a kd-tree and an interval tree (d, bottom right).

lar "spans" $[v_{\min}, v_{\max})$. These spans are represented as points $(v_{\min}, v_{\max}) \in \mathbb{R}^2$ in "*span space*" above its diagonal. A cell is active with respect to an isovalue t if and only if its span lies to the upper left of the point (t,t) in span space, see Figure 1, part (a). When points in span space are organized in a *kd-tree,* active cells can be rapidly extracted, see Livnat, Shen, and Johnson[14] and Figure 1 (b). Without any further checks all cells with points below a node in the tree may be reported as active, respectively empty, if the region corresponding to the kd-tree node in span space is contained in the area to the upper left of (t,t), respectively disjoint from that rectangle.

The per cell space requirements for the kd-tree are given by space for the $v_{\min}$ and $v_{\max}$ values and a pointer back to the cell in addition to storage for the tree structure (unless a pointerless tree is used). For example, in many volume data sets a data value requires 2 bytes, and a pointer typically amounts to 4 bytes. In this case 8 bytes per cell are required, which causes a memory requirement for the kd-tree about 4 times as large as the size of the volume data set itself. For kd-trees, the worst-case search time was shown to be $O(k+\sqrt{n})$, where n again denotes the volume data set size.[13]

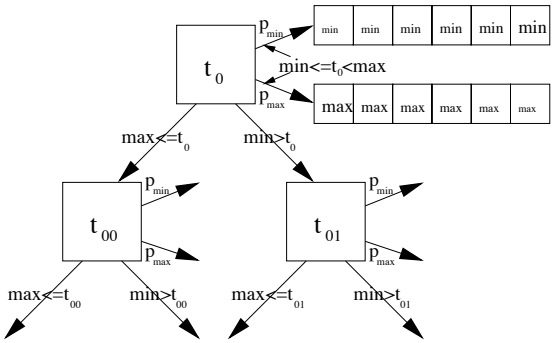


Figure 2: Sketch of the interval tree data structure.

The same authors together with Hansen[19] suggested another way to efficiently store the points in span space by sorting the points into buckets generated by a 2d-lattice and a kind of run length coding, which allows to skip empty buckets during the search.

As an alternative to kd-trees, Cignoni, Marino, Montani, and Puppo [9] proposed the use of *interval trees*. These binary trees are recursively built as follows, compare Figures 1 (c) and 2. Let an arbitrary set of intervals, respectively points in span space, be given. Choose a threshold $t_0 \in \mathbb{R}$. Then the set of intervals can be partitioned into three disjoint subsets, intervals that contain the threshold, intervals below, and intervals above the threshold. The intervals containing the threshold t_0 are associated to the root node of the interval tree. The set of lower intervals are passed on to the left child node and the upper intervals to the right child node. For the latter two sets the procedure is recursively iterated, using suitable thresholds t_{00} and t_{01}. The procedure terminates, when all intervals are attached to a node of the tree. At each node of the interval tree, the associated cells are stored twice using two lists. One list is sorted with increasing corresponding $v_{\min}$ values (called "min-list") and the other list is sorted according to decreasing $v_{\max}$ values (called "max-list"). Thus, the per cell storage amounts to two values $v_{\min}$ and $v_{\max}$ plus two pointers to the cell in addition to the tree structure. Therefore, the interval tree is more expensive in terms of memory than the kd-tree. For the common case of 2 byte values and 4 byte pointers the per cell storage is 12 bytes, i.e., 6 times the amount of storage taken

by the volume data set.

The search for intervals containing the isovalue t starts by visiting the root node. If $t < t_0$, then the corresponding min-list is reported until the first element is detected with $t < v_{\min}$, and the search continues by visiting the left child node recursively. If $t_0 < t$, then the corresponding max-list is reported until the first element is found with $v_{\max} \leq t$ and the search continues by visiting the right child node. If $t = t_0$ then the entire min- or max-list is reported and the search terminates. The computational complexity of the search is $O(k + \log h)$, where k is the output size and h is the number of nodes in the interval tree[9]. With this result, the method is almost output sensitive and with appropriate choice of the thresholds optimal in the sense of Bajaj et al.[5]

In order to reduce the space requirements for the interval or kd-trees, one may restrict to a subset of cells such that all edges in the volume grid are covered[9]. Since a typical edge is shared by four cells only one out of four neighbouring cells needs to be considered. After cell extraction, one propagates the isosurface from the reported active cells on to the neighbouring cells that were not covered in the data structure. Of course, this method of space reduction increases the extraction time because three of four cells have been left out and need to be reconstructed. Although storage can be reduced this way, the method is neither adaptive with respect to a given memory constraint nor optimally designed as the hybrid method in this paper.

Isosurface extraction with kd- or interval trees is efficient, but the enormous memory requirements may be prohibitive when the volume data sets are large.

To overcome the memory bottleneck, Chiang, Silva, and Schroeder[8] devised an out-of-core method based on an interval tree. 'Out-of-core' means, that the auxiliary data structure is stored externally on a disk such that the parts needed for an extraction process can be efficiently read into main memory. One external data block of the structure either contains a group of neighbouring interval tree nodes or a part of a min- or max-list. In a previous out-of-core method of Chiang and Silva[7], a segment tree is stored on the external device. A segment tree is a variation of an interval tree, using nodes with more than two children and containing pointers to so-called multi-lists, as many as one can store in an external data block.

Another method was proposed by Bajaj, Pascucci and Schikore[4] to overcome the problems of memory consumption of auxiliary data structures. It first searches a suitable subset of all cells for active cells from where the isosurface is generated by means of a propagation algorithm. The subset of these seed cells has to be properly selected in order to guarantee that no components of the isosurface are missed. Other versions of the *seed set method* were described by Kreveld et al[12], based on a data skeleton built on the set of local maximum, minimum and saddle points of the given data function; and by Itoh, Yamaguchi, and Koyamada,[11] built by volume reduction while maintaining the topology and all extreme values of the set.

Shen and Johnson[20] proposed a method that is suitable for updating the extracted set of cells when the isovalue slightly changes. Cells are sorted in two lists, once with respect to increasing $v_{\min}$-values and once according to $v_{\max}$-values. With these lists the two sets of cells c with $v_{\min} \leq t$ and $t \leq v_{\max}$ are identified and intersected. Bordoloi and Shen [6] transformed the span space to a space containing the doubled centers $U = v_{\min} + v_{\max}$ and sizes $V = v_{\max} - v_{\min}$ of the intervals. This causes the interval test to turn to the inequation $V > |U - 2t|$. To reduce extraction time, Bordoloi and Shen suggested to use a quantization scheme in the UV-space, which is constructed from the distribution of the given list of intervals.

1.3 The hybrid approach for optimal memory constrained isosurface extraction

Let us compare the performance of a number of basic isosurface extraction methods by means of an example, the XMAS data set.[1] Figure 3 shows a sample isosurface for this CT data set of 16-bit values of resolution $512 \times 512 \times 499$, which amounts to 250 MB of volume data. We explored extraction methods using an octree spatial subdivision, kd-tree and interval tree structures for the span space approach, and out-of-core extraction with an external interval tree, next to the sequential search by enumeration without acceleration. The extraction was performed on a machine with an Athlon processor at 1410 MHz for many uniformly distributed isovalues from the range of the values in the data set. The timings do not include time spent for outputting the extracted cells, because that time is independent of the method. The timings were averaged and reported in Table 1.

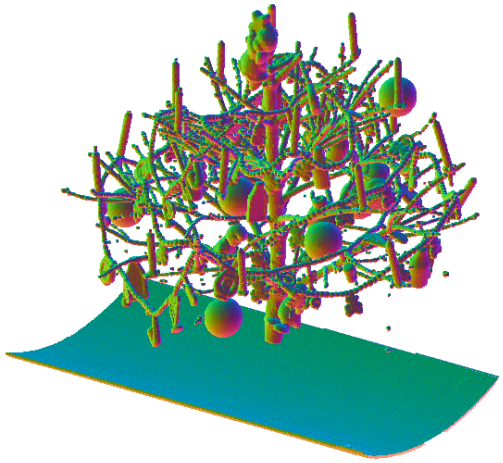


Figure 3: An example isosurface from the $512 \times 512 \times 499$ XMAS data set.

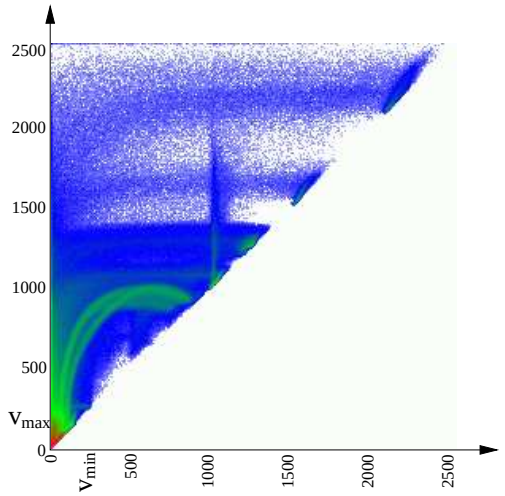


Figure 4: Span space histogram of the XMAS data set.

Method	Search trees (MB)	Total space (MB)	Time (ms)	Speedup factor
enumeration	0	250	5987	1
octree	109	359	87.5	68.4
kd tree	837	1087	3.92	1530
interval tree	1256	1506	2.00	3000
out of core	0	250	17.14	350

Table 1: Results for cell extraction for the CT-scanned XMAS-Tree data set of size $512 \times 512 \times 499$ with 2 bytes per voxel, resulting in 250 MB altogether. The table shows the amount of memory needed for the volume data and the auxiliary data structures for five methods of cell extraction along with the resulting cell extraction times averaged over a sequence of isovalues of equal distance. The out-of-core extraction (using interval trees) results in a speedup without needing any additional main memory space, but it needs 1261 MB of external space and a long preprocessing time to build the data structure. The graph in Figure 5 visualizes these results.

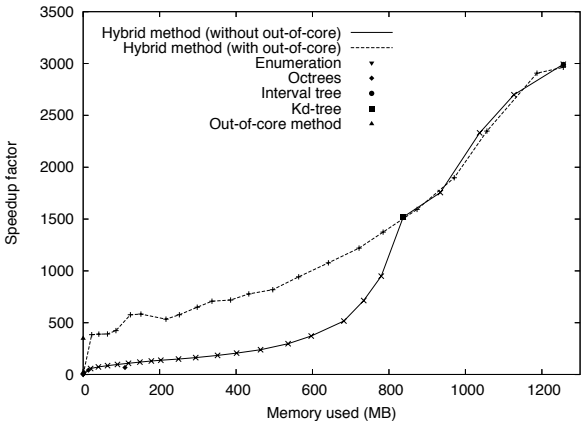


Figure 5: Comparison of methods for cell extraction for the XMAS tree data set, compare Table 1. The graph displays the corresponding speedup factors relative to sequential search as functions of the amount of memory used for the auxiliary data structures.

With the search by enumeration we were able to process 2.18 million cells per second, resulting in an average cell extraction time of 6 seconds. This rather long extraction process was sped up by an impressive factor of 3000 using the interval tree method. However, the interval tree data structure requires 1256 MB of memory for this

achievement, which may not be available on the user's machine. For example, let us assume that only a total of 500 MB memory for the auxiliary data structures may be used. In this case the machine would be able to accomodate the octree method requiring 109 MB and providing an average extraction time of 87.5 milliseconds, while the kd-tree requiring 837 MB would exceed the available memory and can be ruled out due to unacceptable but necessary memory swapping. The question then arises how one may use the remaining 391 MB in the best possible way in order to raise the speedup factor of 68.4 provided by the octree.

The alternative to simply move all memory intensive data structures like kd- and interval trees out-of-core is not a satisfying option. In our example an average extraction time of 17.14 milliseconds was achieved this way (using out-of-core interval trees), which is 4.4 times, resp. 8.6 times longer than with memory resident kd- and interval trees. Thus, it seems promising to use all available cpu memory in order to at least partially reach the rapid extraction times provided by in-core auxiliary data structures.

In this paper we fill the gap between the state-of-the-art methods by designing a memory adaptive hybrid algorithm that optimally adapts to any given amount of available memory. Its performance for the example is given by the solid line in the graph of Figure 5. For the case of 500 MB of available memory, the hybrid approach will result in an average extraction time of only 23 milliseconds, which is less than 30% of the time required without our hybrid method. i.e., using an octree space subdision. Here only combinations of octrees, kd-trees, interval trees and sequential search were allowed. Taking into account additionally the out-of-core method based on external interval trees we achieved an even smaller average extraction time of only 7.3 milliseconds, 12 times smaller than extraction by octrees alone, and 2.35 times smaller than by out-of-core interval trees alone.

In our hybrid approach, we use a binary spatial subdivision. The leaves of the associated binary tree correspond to spatial regions, which can be searched using one out of a pool of several methods. Construction of the binary space partition tree and selection of the search method at the leaf nodes are optimal in the sense that no other feasible configuration with the same or lower memory requirement can provide for a better acceleration. This paper continues and extends the discussion of isosurface extraction in our earlier work.[18] Most notably, besides a more detailed presentation of the general underlying optimization technique and many other improvements appropriate models for execution time estimation for all methods were newly developed. Furthermore, kd-trees and and an out-of core method were integrated in our system and round up the empirical results. In the next section, we formally develop the structure of the hybrid method and its Lagrangian optimization. The methods including spatial subdivision, enumeration, interval trees, kd-trees and an out-of-core method provide the building blocks of our implementation. In Sections 4 and 5, we provide implementation details and discuss the results for some more data sets.

2 Theory

2.1 Definition of the cell extraction problem

Let us assume that we are given samples v_{ijk} of a scalar function $f : \mathbb{R}^3 \to \mathbb{R}$ at integer coordinates from which we can generate an interpolation $\hat{f} : \mathbb{D} \subset \mathbb{R}^3 \to \mathbb{R}$. Thus, $\hat{f}(i,j,k) = f(i,j,k) = v_{ijk}$ for integers i,j,k in certain ranges, say $i_{\min} \le i \le i_{\max}$, $j_{\min} \le j \le j_{\max}$, and $k_{\min} \le k \le k_{\max}$.

For appropriate i,j,k, we define the cells C_{ijk} as explained before and consider the sets of eight corresponding voxel values

$$V_{ijk} = \{v_{i,j,k}, v_{i+1,j,k}, v_{i,j+1,k}, ..., v_{i+1,j+1,k+1}\}$$

and their convex hulls without the maxima

$$I_{ijk} = [\min V_{ijk}, \max V_{ijk}). \tag{1}$$

Now let a threshold $t \in \mathbb{R}$, i.e., an isovalue be given. Then the *cell extraction problem* for the computation of the corresponding isosurface $\hat{f}^{-1}(t)$ is to report the set of indices

$$L = \{(i,j,k) \mid t \in I_{ijk}\}.$$

In a following step (not included in the methods of this work), an isosurface renderer will compute and render $\hat{f}^{-1}(t) \cap C_{ijk}$ for all $(i,j,k) \in L$.

2.2 Definition of the memory constrained optimization problem

Assume that we are given a set of $m > 1$ search methods that solve the above cell extraction problem. For example, these could include the methods of enumeration, interval tree and kd-tree searching. We consider *spatial partitions* $P = \{B_1, \ldots, B_n\}$, $\mathbb{D} = \bigcup_{k=1}^{n} B_k$ of the underlying volume $\mathbb{D}$ into blocks $B_k, k = 1, \ldots, n$. Each block consists of a union of cells, such that each cell belongs to one and only one block B_k. In our hybrid scheme we assign to each of the blocks B_k one of the m search methods, denoted by the number $a_k \in \{1, \ldots, m\}$. For a partition P with n blocks, we thus have a *method assignment vector* $A = (a_1, \ldots, a_n)$. Our *hybrid cell extraction* proceeds by sequentially processing the blocks $B_1, \ldots, B_n$, applying cell extraction method a_k for block B_k.

We assume that we have a mathematical model describing the *cell extraction time* required by any of the m search methods when extracting cells for any block $B \subset \mathbb{D}$ and any given threshold $t \in \mathbb{R}$. We use the notation

$$\mathcal{T}(B, a, t)$$

for cell extraction time where $a \in \{1, \ldots, m\}$ denotes the search method. Likewise we denote by

$$\mathcal{M}(P), \mathcal{M}(B, a), \text{ and } \mathcal{M}(P, A)$$

models for the *space requirements*; $\mathcal{M}(P)$ for storing the definition of the spatial partition, and $\mathcal{M}(B, a)$ for storing the auxiliary data structures required by search method numbered a applied to the block B of cells. Then the overall space requirements $\mathcal{M}(P, A)$ for our hybrid cell extraction method using a partition $P = \{B_1, \ldots, B_n\}$, and method assignment vector $A = (a_1, \ldots, a_n)$ will be

$$\mathcal{M}(P, A) = \mathcal{M}(P) + \sum_{k=1}^{n} \mathcal{M}(B_k, a_k). \tag{2}$$

A global *memory constraint* M on this overall space requirement can be written as $\mathcal{M}(P, A) \leq M$. Details for models of space and time for cell extraction methods will be given in Section 3.

We need one more formal notion for the definition of our optimization problem, namely a probability distribution for the queries of thresholds $t \in \mathbb{R}$. This is necessary because these isovalues are generally not known beforehand. For example, one can simply assume a uniform distribution over the range of the voxel values contained in the entire volume. Another suitable distribution can be based on the histogram of the voxel values in the volume. Using a probability distribution for isovalues $t \in \mathbb{R}$ allows us to work with the expectation of the time requirement $\mathcal{T}(B, a, t)$ for the cell extraction, denoted by $E[\mathcal{T}(B, a, t)]$. The expectation of the total time $\mathcal{T}(P, A, t)$ required for the cell extraction in the entire volume $\mathbb{D}$ using a partition P, $|P| = n$, with method assignment vector A, then is

$$E[\mathcal{T}(P, A, t)] := \sum_{k=1}^{n} E[\mathcal{T}(B_k, a_k, t)].$$

We are now ready to pose the optimization problem for the hybrid cell extraction algorithm.

Definition 1 (Memory constrained optimization problem) For given volume data, m extraction methods, and a global memory constraint M define the configuration set

$$\mathcal{K} = \{(P, A) \mid P \text{ partition}, A \in \{1, ..., m\}^{|P|}\}$$

and compute a configuration $(P^\star, A^\star) \in \mathcal{K}$ that minimizes the expected cell extraction time,

$$(P^\star, A^\star) = \arg \min_{(P,A) \in \mathcal{K}} E[\mathcal{T}(P, A, t)],$$

subject to the constraint

$$\mathcal{M}(P, A) \leq M.$$

The problem above is very hard to solve in its generality. The configuration space $\mathcal{K}$ is huge and unstructured. Partitioning problems can often be shown to be NP-hard. In the next two sections we therefore first restrict to the case of a fixed partition and then solve the problem for a very large set of hierarchical partitions.

2.3 Optimization method for fixed partitions

In this subsection we solve the memory constrained optimization problem of Definition 1 for the case, in which we consider only one partition $P = \{B_1, \ldots, B_n\}$ in the configuration space $\mathcal{K}$. Thus, the task is to compute

$$A^\star = \arg \min_{A \in \{1, \ldots, m\}^n} E[\mathcal{T}(P, A, t)]$$

subject to the memory constraint

$$\mathcal{M}(P,A) \leq M.$$

Using a Lagrangian multiplier $\lambda \geq 0$ we convert the constrained problem to an unconstrained discrete Lagrangian problem.

Definition 2 (Discrete Lagrangian problem) Let P, $|P| = n$, be a partition of $\mathbb{D}$. For the Lagrangian multiplier $\lambda \geq 0$ compute

$$A^\star = \arg \min_{A \in \{1,\ldots,m\}^n} (E[\mathcal{T}(P,A,t)] + \lambda \cdot \mathcal{M}(P,A)).$$

This solution to an unconstrained problem for a given parameter $\lambda \geq 0$ yields a corresponding solution of the constrained problem as stated by the following proposition.

Proposition 1 Let $\lambda \geq 0$ and assume for $A^\star \in \{1,\ldots,m\}^n$

$$A^\star = \arg \min_{A \in \{1,\ldots,m\}^n} E[\mathcal{T}(P,A,t)] + \lambda \cdot \mathcal{M}(P,A). \quad (3)$$

Then $A^\star$ solves the memory constrained optimization problem minimizing $E[\mathcal{T}(P,A,t)]$ subject to $\mathcal{M}(P,A) \leq \mathcal{M}(P,A^\star)$.

The proposition is a special case of a general theorem of Everett[10]. Figure 6 illustrates the proposition and suggests an intuitive understanding of the statement. The space-time diagram contains the point $(\mathcal{M}(P,A), E[\mathcal{T}(P,A,t)])$ for each of the m^n method assignment vectors A. Points on a line with slope $-\lambda$ have equal values of the functional $E[\mathcal{T}(P,A,t)] + \lambda \cdot \mathcal{M}(P,A)$. Geometrically, the optimization in (3) can be seen as moving a line with slope $-\lambda$ upwards until the first point in the graph is reached, which is a vertex on the lower convex hull of the m^n points in the space-time diagram. This point provides the solution $A^\star$. Clearly, no other point in the graph to the left of this point can provide a lower expected extraction time $E[\mathcal{T}(P,A,t)]$. Therefore, $A^\star$ must be optimal with respect to memory constraint $\mathcal{M}(P,A^\star)$.

To solve the constrained optimization problem for a given memory constraint M, one must solve (3) for several values of the parameter λ until a solution with a memory requirement sufficiently close to the given space M is found. Here the method of bisection or the secant method can be applied.

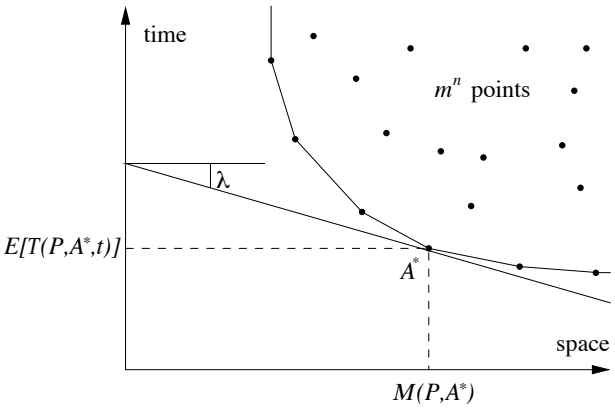


Figure 6: This space-time diagram for the unconstrained optimization contains a point $(\mathcal{M}(P,A), E[\mathcal{T}(P,A,t)])$ for each of the m^n method assignment vectors A.

However, the solution of (3) by enumeration of all m^n points is not feasible, because the complexity of this procedure grows exponentially with the number n of blocks in the partition P. Fortunately, in our case the exponential complexity can be reduced to a linear one by observing a separability property of the problem. To be more precise, let $M' := M - \mathcal{M}(P)$ and assume $M' > 0$. Then by equation (2) we need to find

$$\min_{A \in \{1,\ldots,m\}^n} \sum_{k=1}^{n} E[\mathcal{T}(B_k, a_k, t)]$$

subject to

$$\sum_{k=1}^{n} \mathcal{M}(B_k, a_k) \leq M'.$$

Using these expressions, the optimization in (3) becomes

$$A^\star = \arg \min_{A \in \{1,\ldots,m\}^n} \sum_{k=1}^{n} E[\mathcal{T}(B_k, a_k, t)] + \lambda \cdot \mathcal{M}(B_k, a_k).$$

The term under the sum for index k depends on the method a_k, but not on any of the methods a_i that are used in other blocks $B_i, i \neq k$. Therefore, we can minimize the terms in the sum independently, yielding the solutions

$$a_k = \arg \min_{a \in \{1,\ldots,m\}} E[\mathcal{T}(B_k, a, t)] + \lambda \cdot \mathcal{M}(B_k, a) \quad (4)$$

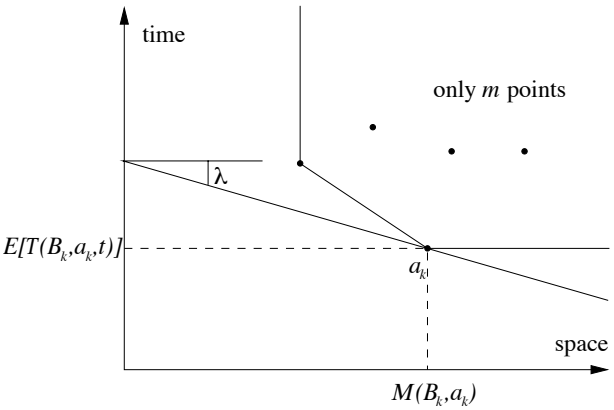


Figure 7: Space-time diagrams for blocks $B_k, k = 1,\ldots,n$. The best method a_k out of m possible ones is searched in each diagram.

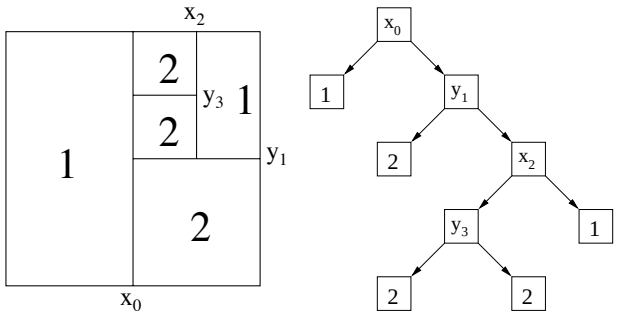


Figure 8: Visualization of a hierarchical partition in the two-dimensional case. One out of two search methods are assigned to the blocks of the partition ($m = 2$). The tree on the right displays the same information.

for $k = 1,\ldots,n$. Figure 7 illustrates the procedure. Instead of searching one point out of m^n points we now search one point for each of the n blocks B_k from a set of only m candidates, thereby reducing the time complexity to a linear one.

2.4 Optimization method for hierarchical partitions

In this subsection we generalize the solution of the memory constrained cell extraction problem allowing several partitions P in the configuration set $\mathcal{K}$. Since the problem with arbitrary partitionings is intractable, we restrict the search to the case of *hierarchical partitions*. These are partitions that are recursively generated as follows. The volume domain $\mathbb{D}$ is partitioned into two or more *macro blocks* consisting of unions of cells as before. Each of these macro blocks can further be subdivided by the same procedure and so on in a recursive manner. The partitioning of $\mathbb{D}$ or of a macro block of $\mathbb{D}$ must follow a deterministic rule. For example, the rule may simply be that a block is split along the largest dimension into two equal subsets, using certain tie breaking rules for the cases, that the block has several longest sides or an odd number of cells along a largest side. A *maximal hierarchical partition* is obtained when the recursive partitioning is carried out until the blocks at the finest level are smaller than some given threshold, that is a parameter of the hybrid cell extraction method.

A hierarchical partition can be identified with a corresponding tree. A node ν corresponds to a spatial volume that is equal to the union of the volumes belonging to the child nodes μ of ν. In particular, the root node corresponds to the entire volume $\mathbb{D}$, the other inner nodes to macro blocks, and finally the leaf nodes correspond to the blocks $B_k, k = 1,\ldots,n$ of the hierarchical partition P. A feasible configuration $(P,A) \in \mathcal{K}$ can be visualized as a spatial partition P whose blocks B_k are labelled by the choices a_k for the corresponding methods. Equivalently, we may draw or store the corresponding tree and label the leaves by numbers a_k, see Figure 8 for an example. We now define the set of allowed partitions to be derived from all pruned subtrees of the maximal hierarchical partition.

Definition 3 (Configuration space) Let a maximal hierarchical partition $P_{\max}$ of $\mathbb{D}$ be given. Then the configuration space for the memory constrained optimization problem in Definition 1 is the set

$$\mathcal{K} = \{(P,A) \mid P \in \mathcal{P} \text{ and } A \in \{1,...,m\}^{|P|}\}$$

where $\mathcal{P}$ consists of all partitions P that can be obtained from $P_{\max}$ by arbitrary pruning of the corresponding partitioning tree.

With the configuration space $\mathcal{K}$ defined for hierarchical

partitions, we generalize the discrete Lagrangian problem from fixed to hierarchical partitions.

Definition 4 (Global Lagrangian optimization problem) Let $P_{\max}$ be the maximal hierarchical partition of $\mathbb{D}$ and let $\mathcal{K}$ be the associated configuration space of hierarchical partitions. The corresponding Lagrangian global cost function is

$$J_\lambda(P,A) := E[\mathcal{T}(P,A,t)] + \lambda \cdot \mathcal{M}(P,A) \tag{5}$$

and the global Lagrangian optimization problem for hierarchical partitions and a Lagrangian multiplier $\lambda \geq 0$ is to compute

$$(P^\star,A^\star) = \arg \min_{(P,A)\in\mathcal{K}} J_\lambda(P,A). \tag{6}$$

As in Proposition 1 the solution $(P^\star,A^\star)$ of the unconstrained global Lagrangian optimization problem is also a solution to the constrained minimization of $E[\mathcal{T}(P,A,t)]$ satisfying $\mathcal{M}(P,A) \leq \mathcal{M}(P^\star,A^\star)$.

The cardinality of the set $\mathcal{P}$ of allowed partitions is very large, being superexponential in the depth of the maximal tree belonging to $P_{\max}$. For example, if $P_{\max}$ corresponds to a complete binary tree of depth $l > 2$, then $|\mathcal{P}|$ is larger than $2^{(2^{l-2})}$. Therefore, to solve the memory constrained cell extraction problem with hierarchical partitions, it is not feasible to use the optimization technique from the last section applied to each partition $P \in \mathcal{P}$.

However, a solution is possible in practise by observing a fundamental principle of optimality in the problem setting. To formulate this principle, we first need to extend the unconstrained Lagrangian space/time optimization problem to all macro blocks corresponding to nodes ν of the maximal partition tree $P_{\max}$. For a given node ν, let us denote by B_ν the corresponding (macro) block of cells and by $\mathcal{P}^{(\nu)}$ the set of all hierarchical partitions of B_ν which are embedded in $\mathcal{P}$. Then the Lagrangian cost function relative to the cell extraction problem at node ν, respectively at (macro) block B_ν, using a partition $P_\nu \in \mathcal{P}^{(\nu)}$ and a method assignment vector $A_\nu \in \{1,\ldots,m\}^{|P_\nu|}$ is naturally defined by

$$J_\lambda(P_\nu,A_\nu) := E[\mathcal{T}(P_\nu,A_\nu,t)] + \lambda \cdot \mathcal{M}(P_\nu,A_\nu).$$

Thus, at all nodes ν of the maximal partitioning tree $P_{\max}$, we can pose a *local Lagrangian optimization problem for cell extraction* in B_ν, namely to compute

$$(P_\nu^\star,A_\nu^\star) = \arg \min_{(P_\nu,A_\nu)\in\mathcal{K}^{(\nu)}} J_\lambda(P_\nu,A_\nu).$$

where

$$\mathcal{K}^{(\nu)} = \{(P_\nu,A_\nu) \mid P_\nu \in \mathcal{P}^{(\nu)} \text{ and } A_\nu \in \{1,...,m\}^{|P_\nu|}\}.$$

Proposition 2 (Principle of optimality) Let $\lambda \geq 0$, $P_{\max}$ the maximal hierarchical partition of $\mathbb{D}$, and $(P^\star,A^\star)$ be the solution of the global Lagrangian optimization problem for cell extraction with hierarchical partitions (Definition 4). Let ν be an inner node of the optimal partition $P^\star$. Then the subtree P_ν rooted at node ν in $P^\star$ together with the method assignment vector A_ν attached to the leaf nodes of P_ν provides the solution $(P_\nu^\star,A_\nu^\star)$ of the local Lagrangian optimization problem relative to node ν.

The proposition is essentially due to the same separability property that we have explained in Subsection 2.3. In order for the global solution $(P^\star,A^\star)$ to be optimal, every partition subtree of $P^\star$ with associated partial method assignment vector must be optimally designed, which means that the corresponding local Lagrangian cost function must be minimal, thereby providing a solution to the local Lagrangian optimization problem. We refrain from writing down a formal proof of this fact.

The principle of optimality in Proposition 2 leads us to a bottom-up dynamic programming solution of the global Lagrangian optimization problem. We can begin by solving the local optimization problems at the leaf nodes, where the local partition is trivial, i.e., consisting of only one single block. Thereafter, the computation of the local solutions at inner nodes are possible by propagating known solutions for the corresponding child nodes up, requiring only a few comparisons of easily computed Lagrangian cost functions. Thus, local solutions are computed from the leaf nodes up all the way to the root node at which point the global solution is obtained.

In terms of formulas the procedure is as follows. For all leaf nodes ν of the maximal tree $P_{\max}$ the solution of the local Lagrangian optimization problem is given by the trivial partition $P_\nu^\star$ (consisting of the single block of cells corresponding to the leaf node ν of $P_{\max}$) and the best method assignment vector $A_\nu^\star = (a)$ with

$$a \quad = \quad \arg \min_{a=1,\cdots,m} J_\lambda(P_\nu^\star,(a))$$

$$= \quad \arg \min_{a=1,\cdots,m} E[\mathcal{T}(B_\nu,a,t)] + \lambda \cdot \mathcal{M}(B_\nu,a).$$

For all inner nodes ν of the maximal tree we let P_ν^0 denote the trivial partition at node ν (only one block, i.e., B_ν itself) and compute

$$\min_{P_\nu,A_\nu} J_\lambda(P_\nu,A_\nu) = \min \left\{ \min_{a=1,\cdots,m} J_\lambda(P_\nu^0,(a)), \right. \tag{7}$$
$$\left. \Delta_{E[\mathcal{T}]}(\nu) + \lambda\Delta_{\mathcal{M}} + \sum_{\mu \text{ child of } \nu} \min_{P_\mu,A_\mu} J_\lambda(P_\mu,A_\mu) \right\}.$$

The first term takes care of the case in which the volume belonging to node ν is not further partitioned and searched using a single fixed method. The second term handles the case in which the volume of node ν is partitioned. To be precise, it contains a small additive constant $\Delta_{E[\mathcal{T}]}(\nu) + \lambda\Delta_{\mathcal{M}}$, where $\Delta_{\mathcal{M}}$ denotes the additional storage space that is needed to indicate in the optimal partition that the volume B_ν at node ν is further partitioned. The term $\Delta_{E[\mathcal{T}]}(\nu)$ reflects the expected time required checking the node ν during cell extraction. By computing local optimal solutions for the tree nodes in a bottom-up fashion, we may assume, that for the child nodes μ we already know the optimal local solutions. Thus, we have direct access to the minimal local Lagrangian costs $J_\lambda(P_\mu,a_\mu)$.

Overall, we have to compare m cost function values at each leaf node and $m+1$ cost function values at each inner node of the maximal partition tree. If we assume m as a constant value, and if the Lagrangian cost $E[\mathcal{T}(B_\nu,a,t)] + \lambda \cdot \mathcal{M}(B_\nu,a)$ can be calculated within $O(n' \log(n'))$ time for the block B_ν of n' cells and a method a, then it can be shown that building the optimal tree costs $O(n \log^2 n)$ time (here n is the number of cells in the given volume) for each Lagrange multiplier λ.

2.5 Summary of the optimization method

Overall, the proposed solution for the optimal memory constrained cell extraction problem proceeds by preprocessing the volume data solving the Lagrangian optimization problem for a suitable set of parameters $\lambda \geq 0$. The optimal solutions together with the corresponding space requirements are then available for multiple cell extraction passes obeying an arbitrary memory constraint for the required auxiliary data structures. The preprocessing needs to be executed only once and can be carried out offline. The results can be stored to a file and can be used multiple times for different memory constraints. In more detail, the steps are as follows:

Algorithm 1 (Preprocessing)

1. *Read volume data.*
2. *Build full partitioning tree $P_{\max}$ using a minimal block size (e.g., $2 \times 2 \times 2$) for the termination criterion at the leaf nodes.*
3. *For all nodes ν and methods $a = 1,...,m$, estimate the required space $\mathcal{M}(B_\nu,a)$ and the expected extraction time $E[\mathcal{T}(B_\nu,a,t)]$.*
4. *Choose a sequence (λ_i) with $0 < \lambda_1 < \lambda_2 < \cdots < \lambda_l$. Also choose a resolution parameter $\delta > 0$.*
5. *For each λ_i, run the bottom-up Lagrangian optimization and obtain the optimal partition $P_i^\star$, the assignment vector $A_i^\star$ and the required space $\mathcal{M}(P_i^\star,A_i^\star)$.*
6. *Insert more λ-values and go to Step 5 to ensure that $\mathcal{M}(P_i^\star,A_i^\star)/\mathcal{M}(P_{i+1}^\star,A_{i+1}^\star) \leq 1+\delta$ for all i.*

After preprocessing, the cell extraction and the isosurface rendering commence as follows:

Algorithm 2 (Cell extraction and rendering)

1. *Read volume data.*
2. *Determine available space M for auxiliary data structures.*
3. *Extract optimal partition $P_{i(M)}^\star$ and method assignment vector $A_{i(M)}^\star$ from the preprocessing, i.e., compute $i(M) = \min\{i = 1,\ldots,l \mid \mathcal{M}(P_i^\star,A_i^\star) \leq M\}$.*
4. *Build auxiliary data structures as given by $P_{i(M)}^\star$ and $A_{i(M)}^\star$.*
5. *User input: isovalue t.*
6. *Extract the cells using the methods and data structures generated in Step 4.*
7. *Produce the polygonal approximation of the isosurface in each extracted cell and display its graphic rendering (or output the result in another way).*
8. *If desired go to Step 5.*

2.6 Optimization with multiple constraints

The original theorem of Everett on discrete Lagrangian optimization considers considers minimization of a generalized expression like

$$E[\mathcal{T}(P,A,t)] + \sum_{i=1}^{l} \lambda_i \mathcal{M}_i(P,A).$$

The theorem states that the solution $(P^\star,A^\star)$ minimizing this expression also solves the problem of minimizing $E[\mathcal{T}(P,A,t)]$ subject to $\mathcal{M}_i(P,A) \leq \mathcal{M}_i(P^\star,A^\star)$ for $i = 1,\ldots,l$. Thus, this generalization may be used for the use of several types of memory of different size and speed. However, the multi-dimensional search for the corresponding parameter values $(\lambda_1,\ldots,\lambda_l)$ that yields the solution for a given set of memory constraints, is more difficult than the simple onedimensional case which can be solved by bisection.

The optimization with multiple Lagrangian parameters may be appropriate when also an out-of-core method is considered as a basic method for cell extraction at the leaf nodes of the binary space partition of the volume data. For example, one may use externally stored interval trees, as described by Chiang, Silva and Schroeder[8], storing subgroups of interval tree nodes and parts of the min- and max-lists in external data blocks. In such a case, one may pose either an external memory contraint, or just introduce an external memory factor $\mathcal{H}$, giving the cost of one byte of external memory compared to one byte of main memory (formally $\mathcal{H} = \frac{\lambda_2}{\lambda_1}$). A choice of a large value for $\mathcal{H}$ like $\mathcal{H} \approx 1$ favors in-core methods, while a value of $\mathcal{H}$ near 0 indicates that out-of-core memory essentially is free, yielding frequent use of out-of-core extraction methods when main memory is not sufficient to host the best auxiliary data structures for cell extraction.

3 Models for space and time

Optimizing the expected cell extraction time as described in the previous section requires a mathematical model of query values issued by the user in the form of a probability distribution. Moreover, models of space and expected extraction time for the basic cell extraction methods are necessary. These models provide the interface between the general optimization method and the particular basic cell extraction methods used at the leaves of the partitioning tree. in this section we develop these models by analysing the four main extraction methods ($m = 4$), listed in the following table.

Number (a)	Method	Memory	Speed
1	enumeration	low	slow
2	interval tree	high	fast
3	kd-tree	high	fast
4	out-of-core	(*)	medium

(*) The out-of-core method uses low main memory but much external memory.

The space requirements are zero for enumeration ($a = 1$). For interval trees ($a = 2$), they can be readily measured without actually building the min- and max-lists for the interval trees. For kd.trees ($a = 3$), we only have to count the 'nodes' of the (pointerless) kd-tree, i. e., the size of the array determining it. Forout-of-core interval trees ($a = 4$), the main memory used is almost zero (only one pointer to the out-of-core file is needed). The out-of-core memory can be calculated by counting or calculating the data blocks used, which can be directly done from the evaluation of the in-core interval tree ($a = 2$).

It is more difficult to estimate the time requirements for cell extraction at the nodes of the maximal partition tree. Measuring cell extraction times empirically is not a feasible option due to complexity reasons. Therefore, in the next four subsections we set up mathematical models for the extraction times which can be rapidly evaluated in Step 3 of the preprocessing phase of the optimization.

We begin by introducing notation. Let us denote by $B_\nu = [i_0,i_1] \times [j_0,j_1] \times [k_0,k_1]$ the volume that is attached to node ν of the partition tree. Then $|B_\nu| = (i_1 - i_0)(j_1 - j_0)(k_1 - k_0)$ is the number of cells contained in B_ν. Furthermore, we denote by $I_\nu = [\min V_\nu, \max V_\nu)$ the interval of voxel values belonging to the block B_ν, where

$$V_\nu = \{v_{ijk} \mid i_0 \leq i \leq i_1, j_0 \leq j \leq j_1, k_0 \leq k \leq k_1\}.$$

For any interval I, let us denote by $Pr(I)$ the probability that the query value t is in the interval, $t \in I$.

3.1 Enumeration ($a = 1$)

In the enumeration method, cells in the block B_ν belonging to the node ν of the partition tree are processed layer

by layer, and row by row from left to right in each layer. For a given cell one computes the value range for the cell's four left voxels, i.e., the (half-open) convex hull of the four voxel values. Then one sets a ternary flag indicating whether the isovalue t is below (-1), in (0), or above (1) the range. The same procedure is carried out for the cell's four right voxels. Then it follows, that the cell is active relative to the isovalue t, if and only if both flags are zero, or the flags have opposite sign. The flag for the four right voxels is equal to the left flag of the following cell to the right, and, thus does not need to be recomputed. Thus, the computational load is identical for all cells in the block, except for the first cell in each row which requires about twice as many computations. For our purposes it suffices to simply model the compute time for the enumeration method to be proportional to the number of cells in the block, denoted by $|B_v|$. Therefore,

$$E[\mathcal{T}(B_v,1,t)] = \alpha_1 Pr(I_v)|B_v| \tag{8}$$

where α_1 is a machine dependent parameter that needs to be determined empirically, see Subsection 3.5.

3.2 Interval trees ($a = 2$)

A simple model for the expected cell extraction time of the interval tree method can be based on the nearly output sensitive character of the method, stating that the time is proportional to the expected number of cells reported, $\sum_{C_{ijk}\subset B_v} Pr(I_{ijk})$. Thus, with a suitable constant α_2 we suggest

$$E[\mathcal{T}(B_v,2,t)] = \sum_{C_{ijk}\subset B_v} \alpha_2 Pr(I_{ijk}). \tag{9}$$

A more accurate model would also take into account the processing of the interval tree besides the scan of the min- and max-lists of the tree. To this end one would add a term $\sum_\eta Pr(I_\eta)\cdot\alpha_2'$, where I_η denotes the interval of isovalues which will lead the search algorithm to examine a node η in the interval tree. These intervals can be obtained recursively. In our experiments, however, we found that the contribution of this extra term can safely be neglected. Thus, in our system we use only equation 9.

3.3 Kd-trees ($a = 3$)

For isosurface extraction with kd-trees we apply the pointerless kd-tree data structure and the corresponding optimized search as given by Livnat, Shen, and Johnson[14]. For space limitations we cannot review this material here and must refer to the cited paper.

Each node η in the kd-tree corresponds to a rectangular region in span space, $I_{1\eta}\times I_{2\eta} = [\min_l,\min_h]\times[\max_l,\max_h]$ which contains a corresponding interval (i.e., point in span space) in addtion to the intervals of all nodes below η. These regions can be computed recursively top-down, beginning with the root node for which the region is the square ranging from the minimum to the maximum values of the data set in each of the two dimensions. We have

$$I_{1\eta_l}\times I_{2\eta_l} := \begin{cases} (I_{1\eta}\cap(-\infty,\min_\eta])\times I_{2\eta} \\ \text{if } \eta \text{ splits the intervals by minimum} \\ I_{1\eta}\times(I_{2\eta}\cap(-\infty,\max_\eta]) \quad \text{otherwise} \end{cases}$$

$$I_{1\eta_r}\times I_{2\eta_r} := \begin{cases} (I_{1\eta}\cap[\min_\eta,\infty))\times I_{2\eta} \\ \text{if } \eta \text{ splits the intervals by minimum} \\ I_{1\eta}\times(I_{2\eta}\cap[\max_\eta,\infty)) \quad \text{otherwise} \end{cases}$$

The time required by an isovalue query of the kd-tree is dominated by the comparisons of the isovalue with the min and max values of the intervals of the cells. Thus, for each cell, respectively node of the kd-tree, either 0, 1, or perhaps 2 comparisons may occur. For the analysis of the expected extraction time we therefore must count the number of nodes for these three cases.

Let $[\min_l,\min_h]\times[\max_l,\max_h]$ be the region associated to a kd-tree node and assume the case that $\min_h < \max_l$, see Figure 9 (top). If the isovalue $t \notin [\min_l,\max_h)$, then the tree node is pruned in the kd-tree traversal requiring no compute time. If the isovalue $t \in [\min_h,\max_l)$, then the interval contained in the node must contain the isovalue t, thus, corresponds to an active cell. In this case no comparision of t with either the min or max value of the interval is necessary. One of the limits min and max of the has to be tested, if t is contained in one of the outer intervals $[\max_l,\max_h)$ and $[\min_l,\min_h)$.

In the other case, i.e., if $\min_h \geq \max_l$, see Figure 9 (bottom) for an example, the isovalue t may have to be compared with both ends of the interval, min and max, if $\max(\min_l,\max_l) \leq t < \min(\min_h,\max_h)$. Overall, with the convention $Pr([a,b]) = 0$ for $b < a$, we arrive at the

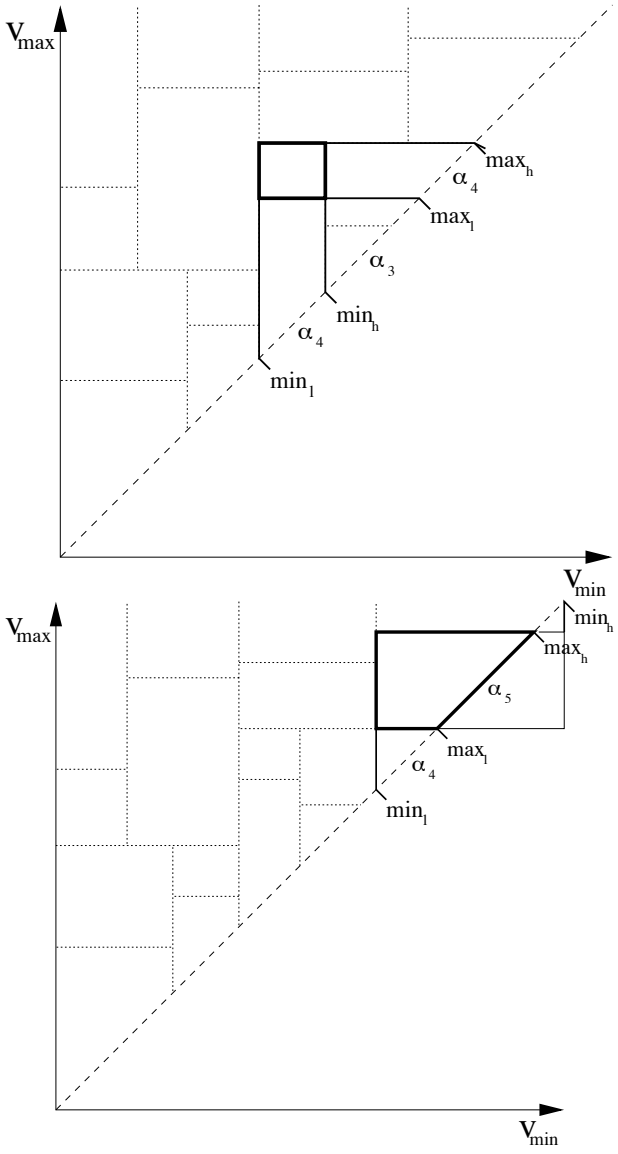


Figure 9: Span space representation of two cases of the kd-tree node evaluation. In the second case, the upper part of α_4 will not occur because $\min_h > \max_h$ and thus, $[\min_h, \max_h)$ is empty.

expected extraction time

$$T_\eta = \begin{cases} Pr([\min_h, \max_l]) \cdot \alpha_3 + \\ [Pr([\min_l, \min_h]) + Pr([\max_l, \max_h])] \cdot \alpha_4 \\ \text{for } \min_h < \max_l \\ Pr([\max(\min_l, \max_l), \min(\min_h, \max_h)]) \cdot \alpha_5 + \\ [Pr([\min_l, \max_l]) + Pr([\min_h, \max_h])] \cdot \alpha_4 \\ \text{otherwise.} \end{cases} \quad (10)$$

Here, α_3, α_4, and α_5 are machine dependent parameters. Finally, the expected extraction times are added by

$$E[\mathcal{T}(B_v, 3, t)] = \sum_\eta T_\eta$$

where the sum is taken over all nodes in the kd-tree.

For the computation of $E[\mathcal{T}(B_v, 3, t)]$ according to the above method, it is required to build and traverse the entire kd-tree for the block B_v of cells. Since in the preprocessing, thousands and even millions of such nodes may need to be processed, large compute times may be required for obtaining the expected extraction times with kd-trees. In order to reduce these long compute times we suggest a probabilistic approximation by choosing a random subset of the given cells in B_v. With this smaller set of cells, the kd-tree of associated intervals is built and processed as explained above. The results are properly scaled or modified to estimate the expected extraction times for the full kd-tree.

We discuss details only for the case of a node from the small kd-tree corresponding to a rectangle $R = [min_l, min_h] \times [max_l, max_h]$ with $(min_h < max_l)$, i.e., the first case in (10) and the upper part of Figure 9, the other case being similar. We aim at replacing the term

$$Pr([min_l, min_h]) + Pr([max_l, max_h]) \cdot \alpha_4$$

by an appropriate one which should estimate the corresponding quantity in the original large kd-tree. If the random downsampling of the cells in B_v occurs at a rate of $\rho > 1$ (reducing the number of cells by a factor of $1/\rho$), we simply assume that each interval in the small kd-tree corresponds to ρ intervals in the original kd-tree which are uniformly distributed in the corresponding rectangle R. Thus, the rectangel R corresponds to ρ smaller rectangles in the original kd-tree, and we arrive at

$$\sqrt{\rho} \cdot [Pr([min_l, min_h]) + Pr([max_l, max_h])] \cdot \alpha_4.$$

Next we need to replace the term $Pr([min_h, max_l]) \cdot \alpha_3$ by an estimate valid when replacing R by about ρ smaller rectangles. Here we take ρ times the value obtained for a small rectangle located near the center (c_{min}, c_{max}) of R. Thus, the min_h-value of the small rectangle is $c_{min} + \frac{1}{2\rho}(min_h - min_l)$, and max_l becomes $c_{max} - \frac{1}{2\rho}(max_h - max_l)$.

Note, that when taking c_{min} as the arithmetic mean of min_l and min_h and c_{max} as the mean of max_l and max_h and setting $\rho = 1$, i.e., we do nor subsample the data set, then the resulting approximation is exact. Since the intervals contained in R are not uniformly distributed in practise (see Figures 4 and 17-20), but occur more frequently near the diagonal in span space, taking the geometric mean for $c_{min} = \sqrt{min_l \cdot min_h}$ and $c_{max} = \sqrt{max_l \cdot max_h}$ is better than using the arithmetic means.

We evaluated the procedure using a sequence of volume data sets ranging from 1 to 250 MB. For each of these data sets one large kd-tree was generated. We compared the approximate results based on randomly subsampled data sets with those from the full kd-trees as computed using (10). Figure 10 shows the results of this experiment. The error of the estimated expected extraction times typically was less than five percent. For the downsampling rate we found that reducing the number of cells to $250\sqrt[3]{\#\text{ cells}}$ intervals gave the best results as shown in the figure. The probabilistic estimation reduced the compute time by a factor betwen 20 and 100, where the large savings occur for the large data sets.

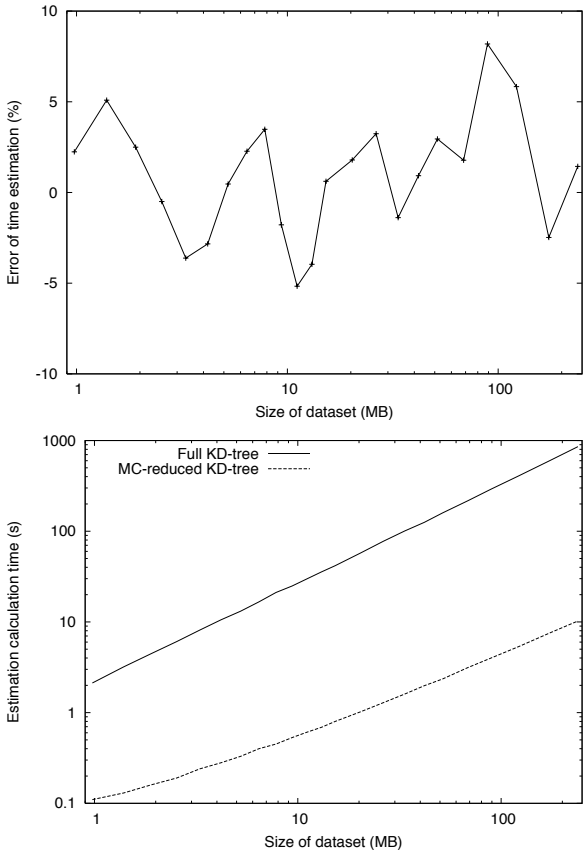


Figure 10: Comparison of the results of original kd-tree extraction time evaluation to an estimation by smaller kd-trees. The second diagram compares the compute times used for the evaluations.

3.4 Out-of-core extraction ($a = 4$)

The overall isosurface extraction time when using an out-of-core interval tree naturally is equal to the time required when using in-core interval tree in addition to time needed to read in disk blocks containing the necessary parts of the interval tree data structure together with disk blocks for the min- and max-lists. For each disk block B_i we can derive an interval $I(B_i)$ of isovalues for which the given disk block must be loaded from disk. Thus, if α_6 denotes the time for a read operation for one disk block, we obtain a total expected time for all disk block read operations as follows

$$\alpha_6 \sum_{i=1}^{N_B} Pr(I(B_i)) \tag{11}$$

where N_B is the total number of disk blocks for the external interval tree and its associated min- and max-lists. We proceed by giving details for the computation of the intervals $I(B_i)$ for the disk blocks B_i that contain a part of the interval tree, and by approximating the partial sum for the disk blocks holding the min- and max lists of the interval tree.

We introduce a partitioning scheme for the nodes of the interval tree such that each region of the tree can be stored in one disk block. Let us assume, that one disk block can store the information of at most $2^{K_1+1} - 1$ interval tree nodes, contained in a binary subtree of depth K_1. The tree partitioning proceeds with a labeling $\ell(\eta) \in \{0, ..., K_1\}$ of the tree nodes η

$$\ell(\eta) := \begin{cases} K_1 & \text{if } \eta \text{ is a leaf} \\ \ell'(\eta) & \text{if } \eta \text{ is not a leaf and not the root} \\ 0 & \text{if } \eta \text{ is the root} \end{cases}$$

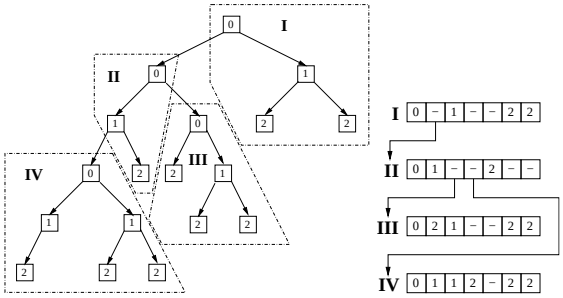


Figure 11: This diagram schematically shows the labeling of interval tree nodes and how disk blocks are defined for external interval tree storage.

where

$$\ell'(\eta) \quad := \quad \min(\{K_1\} \cup \{\ell(\eta') - 1 \mid \eta' \in \{\eta_l, \eta_r\} \text{ exists and } \ell(\eta') > 0\}.$$

and η_l, η_r denote the left and right child nodes belonging to η. In practise the labeling is defined for the tree nodes in bottom-up manner, starting with labels K_1 at all leaf nodes. Figure 11 illustrates an example. With this labeling we define the content of a disk block to be a subtree rooted at any node labeled 0 and reaching down up to K_1 levels and not containing another node labeled 0, see Figure 11.

We now attach intervals I_η to all interval tree nodes indicating the range of isovalues for which a given node η must be inspected in the tree traversal during cell extraction. The intervals I_η are generated in a top-down tree traversal. Initially, for the root node η_{root} of an interval tree attached to the node ν of the partition tree, we define $I_{\eta_{\text{root}}} := I_\nu$, i.e., $I_{\eta_{\text{root}}}$ is the range of the volume data in the block of cells B_ν. If t_0 denotes the threshold at the root node in the interval tree, then the interval of the left child node η_l is $I_{\eta_l} = I_{\eta_{\text{root}}} \cap (-\infty, t_0)$, and for the right child node, it is $I_{\eta_r} = I_{\eta_{\text{root}}} \cap [t_0, \infty)$. The procedure is iterated correspondingly for the child nodes of η_l, η_r and so forth.

With these definitions we have that a disk block containing interval tree nodes must be read in if and only if the isovalue is contained in the interval I_η of the first node η the disk block, i.e., the block with label $\ell(\eta) = 0$. Thus, the expected time to read in the necessary disk blocks is given by

$$T_1 = \alpha_6 \sum_{\ell(\eta)=0} Pr(I_\eta)$$

where α_6 is the time needed to read one disk block.

The min- and max-lists are linearly written into disk blocks, where the last block for each list may not be completely filled. It is possible to derive relevant isovalue intervals for each such disk block in order to evaluate equation (11). However, computing these intervals would require actually building the min- and max-lists, which should be avoided to maintain low computational complexity. Instead we propose to estimate the expected time to read the necessary disk blocks. Let us assume that one disk block can hold up to K_2 min- or max-list entries. Moreover, we may estimate that in the last used disk block of a list from which cells are reported to the output on average only about half of the entries are active. For an isovalue t either the min- or the max-list of an interval tree node η must be processed, if and only if $t \in I_\eta$. Thus, we arrive at an estimated expected time for reading in the disk blocks containing the min- and max-lists as follows:

$$T_2 \approx \alpha_6 \left[\sum_{I \text{ cell interval}} \frac{Pr(I)}{K_2} + \frac{1}{2} \sum_{\eta} Pr(I_\eta) \right].$$

The overall expected cell extraction time for the method with out-of-core interval trees can be calculated by

$$E[T(B_\nu, 4, t)] = E[T(B_\nu, 2, t)] + T_1 + T_2.$$

We remark, that for the computation of $E[T(B_\nu, 4, t)]$ it is sufficient to have available the interval tree structure without having to set up the corresponding min- and max-lists.

We need one more constant in the model, namely for the expected time spent checking an inner node, $\Delta_{E[\mathcal{T}]}(\nu)$, in equation (7), simply estimated as

$$\Delta_{E[\mathcal{T}]}(\nu) = Pr(I_\nu) \cdot \alpha_7.$$

3.5 Parameter estimation

The models for expected extraction times contain unknown parameters $\alpha_1, \ldots, \alpha_6$, which can be determined empirically. To do this, one may generate several partitioning trees P_i with method assignment vectors A_i. It

does not matter, whether they are optimal. In each case i, the mathematical model predicts a run time $\tilde{T}_i$ as a linear combination of the parameters $\alpha_1, \ldots, \alpha_6$. The coefficients are given by the formulas of the last subsection and can be computed. Then, for each i, the actual extraction times are measured and averaged for a set of query values t obeying the underlying probability distribution, yielding empirical expected times T_i. Now the parameters can be determined using standard least squares minimization to fit the model to the data. The Figures 12 and 13 show the quality of this parameter estimation, comparing the modelled extraction time to the measured extraction time for the given methods. The values of the constants will be given in the result section.

4 Implementation

The hybrid algorithm has been presented in the previous sections. Here, we provide details for our choices that we made for our implementation, following the steps listed in Algorithms 1 and 2. We apply the hybrid scheme using four methods ($m = 4$): the enumeration method, the interval trees, the kd-trees, and the out-of-core method with interval trees.

The required space and expected extraction times are computed using a recursive depth-first tree traversal of the maximal partitioning tree (Step 3 of the preprocessing algorithm). In this way, information collected and analyzed at all nodes below the root is passed on to their respective parents, which is used to accelerate the calculations pertaining to the parent nodes. We remark that for estimating space and expected extraction times with the interval tree method no min- and max-lists for interval trees need to be built. It suffices to know the nodes of the trees, which can be derived from a small subset of the intervals that are maintained in the tree.

The resulting values of space requirements and expected times are stored in a table providing random access for the following optimization passes for the various parameters λ_i (Step 5 of the preprocessing algorithm). These bottom-up Lagrangian optimizations are also computed using the recursive depth-first tree traversal algorithm. The resulting optimal hierarchical partitions $P_i^\star$ are stored along with the assignment vectors $A_i^\star$ requiring one symbol per node, where the symbol is from an alphabet of size $m+1$ (one of these symbols denoting an inner node, the other m symbols indicating the method for a leaf node).

Our implementation is in the C language. We remark, that we store the volume data in a linear array instead of a standard three-dimensional array, which saves storage of a 2D pointer array allowing more space for auxiliary data structures.

5 Results

In this section we report on the empirical results using our implementation of the hybrid isosurface extraction method of this paper. We begin by discussing the results of the machine parameter estimation that we performed as explained in Section 3. In the following subsection we discuss the computing time requirements for the offline preprocessing (Algorithm 1). Next we list and characterize our test data sets. Finally, we address the performance of the extraction method for the example data sets and varying memory constraints.

5.1 Machine parameter estimation

For our experiments we used a computer with an AMD Athlon(TM) XP 1600+ processor which is running at 1410 MHz under a Linux operating system. The main memory size of the machine is one gigabyte.

Our results for the estimation of the time constants in Subsection 3.5 yielded the following values in nano seconds.

Parameter	Value 10^{-9} s	Relevance for time in
α_1	42.5	single enumerated cell
α_2	4.84	reading a cell from min-, max-list
α_3	4.04	reading a cell below kd-tree node
α_4	37.6	reading a cell and test of min or max
α_5	0	reading a cell and test of min and max
α_6	2287	out-of-core block read
α_7	135.7	inner node of partition tree

Table 2: Machine constants for AMD Athlon XP 1600+.

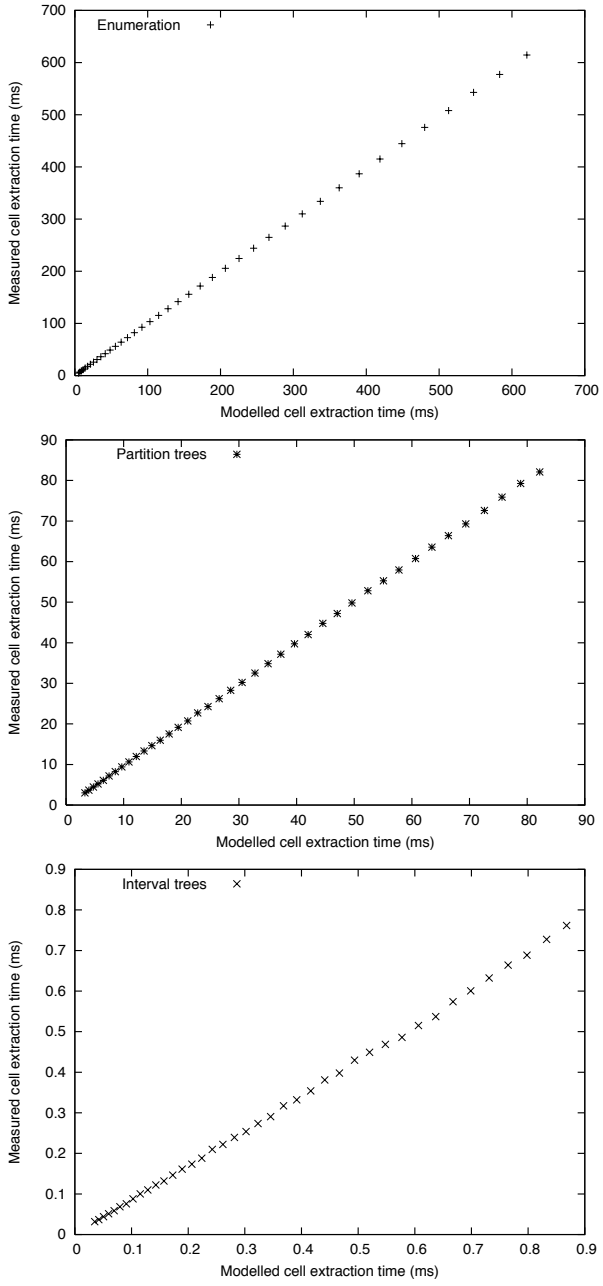


Figure 12: Modelled expected cell extraction time versus actual cell extraction time averaged over different isovalues, shown for the implemented methods enumeration, partition trees and interval trees.

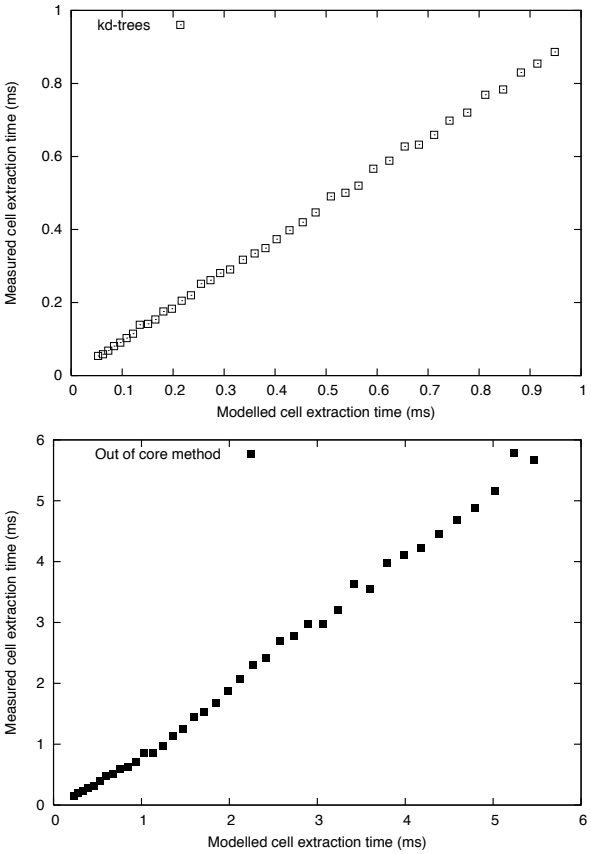


Figure 13: Modelled expected cell extraction time versus actual cell extraction time for the methods KD-trees and out-of-core extraction.

The constant α_5 was arbitrarily set to zero because their numerical least squares estimation failed due to lack of sufficient data points. Compared to other operations the computations associated to these constants occur rarely and, thus, setting the estimated times to zero does not incurr a significant loss in overall precision. Note, that the value $\alpha_6 = 2287$ is very high compared to the others, as we could expect for the time of an out-of-core block reading operation. The disk block size in our experimental setup was 512 bytes, capable of holding interval subtrees of depth $K_1 = 3$ (i.e., 15 nodes per block) or $K_2 = 85$ entries of min- or max-lists.

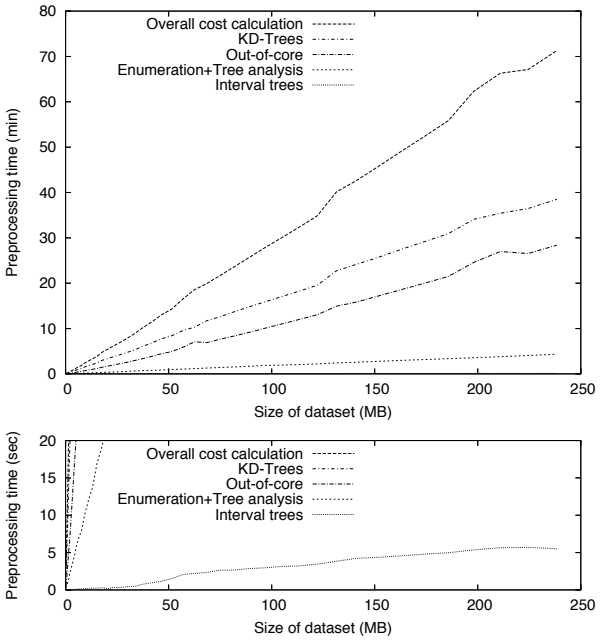


Figure 14: Cost calculation time in the preprocessing for different extraction methods.

We tested the model for expected extraction times for a volume data set using a range of different hybrid trees. In each case, the modelled expected cell extraction time was compared to the empirical time averaged over query values drawn from the corresponding uniform distribution of queries. The results are shown in Figure 12 and 13, demonstrating an accuracy of our model that is sufficient for our application.

5.2 Preprocessing

In this subsection we report on the preprocessing times for a volume data set that is scalable in size (TESTSET[1]). For a desired variable resolution up to $500\times500\times500$ we sampled an algebraic function on a corresponding regular grid, added Gaussian noise to the values, and scaled and quantized the results to 2 byte integers. Thereby we obtained variable size volume data sets up to 238.4 MB.

For the preprocessing we considered $m = 4$ methods (enumeration, interval trees, kd-trees, and the out-of-core

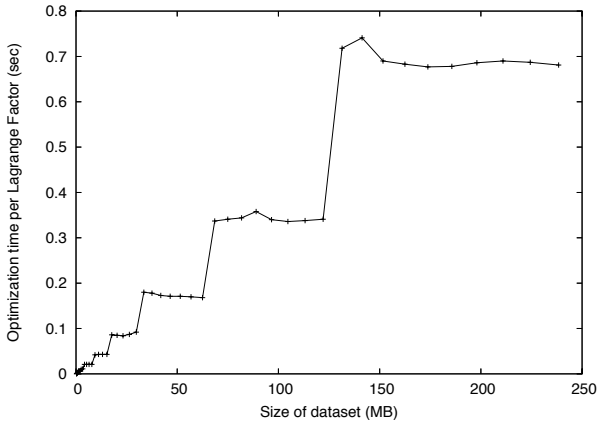


Figure 15: Optimization time per Lagrange factor λ for a series of variable size data sets.

method). For the full size test data set of 250 MB the first three steps of the proprocessing (reading the volume data, building the partition tree down to minimal block size 45 cells, and estimating the space and time required for all four methods at each of the nodes of the partitioning tree amounted to 75 minutes. The time taken for calculating the memory and expected time needed for all methods at all nodes is $O(n\log^2 n)$, thus, roughly linear in practise. Here the rate is about 19 seconds per megabyte. Figure 14 shows the times required to compute the memory and expected time needed for the different methods at all nodes. Of the the total time for any given data set size about 54% was taken by the calculations for the kd-tree, 40% for the out-of-core method based on external interval trees, 6% for setting up the partition tree and computing the intervals attached to the nodes, and only 0.13% for the interval tree.

The following Lagrangian optimization required less than one second for each Lagrange parameter λ. Figure 15 shows these preprocessing times as the size of the volume data set changes. The step like charfacteristic of the curve is due to the construction of the underlying binary space partitions.

The output of the preprocessing contains for each value of the parameter λ the optimal spatial partitioning tree of the volume data and the corresponding interval trees (without the min- and max-lists). For the example of the

Name	Resolution (voxels)	Voxel (bits)	Data Range	Data (MB)
XMAS	512×512×499	16	0–4089	249.5
BUNNY	360×512×512	16	21–63536	180.0
BRAIN	384×400×276	16	0–65280	80.9
BRAIN+	767×799×551	16	0–65280	644.1
BIGHEAD	256×256×225	16	2304–65280	28.0
ENGINE	256×256×110	16	0–65280	13.7
TESTDATA	500×500×500	16	0–65280	238.4

Table 3: Example data sets used. TESTDATA was also used with lower resolutions.

XMAS data set with 93 λ-values a total file size of 1270 kB resulted.

5.3 Example data sets

Table 5.3 summarized the characteristics of some of the data sets we used in our experiments. Except for the data sets BRAIN and BRAIN+ they are available on the internet.[1] The large BRAIN+ data set was upsampled by linear interpolation from the original 384×400×276 BRAIN volume data set by a factor of 2 in each spatial dimemsion. In addition, small additive Gaussian noise was included at the upsampled positions. Figures 3 and 17 to 20 include renderings of example isosurfaces and histograms for the distribution of the interval lengths of the cells. The value region is normed to the interval $[0,1]$ and the interval length frequencies are scaled, such that their integral over the whole value region is 1.

We see that for the medical data set, all interval lengths occur, with a decreasing frequency for increasing length. For the ENGINE and BUNNY data sets, many intervals of large lengths occur near the surface of the objects were cells include object matter as well as surrounding air. The XMAS data set only contains small intervals.

The rendering of isosurfaces (except for XMAS and BRAIN which were raytraced) were obtained by an interactive real-time program, displaying extracted cells as OpenGL points. The (r,g,b) color of such a point was defined using an estimated surface normal vector $v = (v_x, v_y, v_z)$ as $r = \frac{v_x+1}{2}$, $g = \frac{v_y+1}{2}$ and $b = \frac{v_z+1}{2}$.

5.4 Cell extraction

For the data sets BUNNY, BRAIN, BIGHEAD, and ENGINE Figures 17 to 20 show corresponding renderings of example isosurfaces, histograms of cell ranges, and memory-speedup diagrams for the hybrid method, using enumeration, kd-trees, interval trees out-of-core interval trees.In the memory speedup diagrams, the speedup factor is taken to be relative to the speed of the extraction by enumeration (marching cubes style, run time given in captions). In each of these diagram markers show the memory requirements and speedup factors for these four individual methods. For our optimal hybrid method we show two curves. For the lower curve out-of-core extraction was not allowed as one of the methods in the hybrid approach. In the timings we do not include time for rendering, or even reporting the active cells, because these are irrelevant regarding the evaluation of the cell extraction method. The underlying probability distribution for the isovalues was taken to be uniform on the range intervals for the entire data set, i.e., the intervals given in Table 5.3.

The results here and those for the data set XMAS listed already in Table 1Ê show, that the hybrid method is effective as expected. Increasing available memory for the hybrid data structures speeds up the isosurface extraction process. As derived in Section 2 the extraction is optimal in the sense that no other feasible binary space partition and method assignment vector would provide for a higher acceleration of the extraction process.

We note in some details that the empirical memory-speedup curves for the hybrid method are not strictly monotonically increasing as claimed. This is most visible in the jaggy upper memory-speedup curve for BUNNY in Figure 17. This is due to the facts that in the optimization algorithm expected extraction times are considered, while in our measurement we sampled the probability distribution, and, moreover, we can only estimate the expected extraction times, rather than precisely calculate them. This is also the reason that in the speedup diagram of Figure 18 the pure interval tree is not reached (the estimated expected time for the kd-tree method was lower than that for the interval tree method).

The Figure 16 shows the memory-speedup diagram for the data set BRAIN+, which is the large upsampled version of the BRAIN data set (Figure 18). Here we report

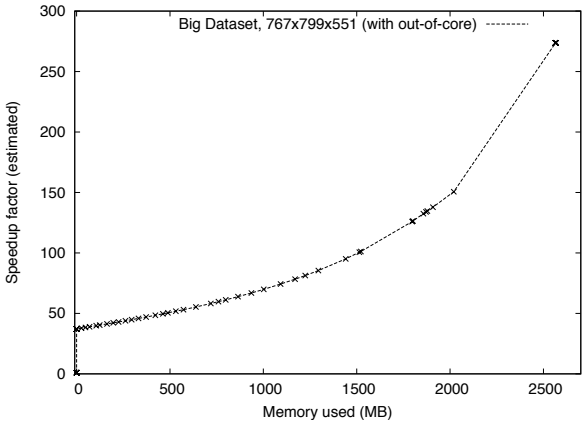


Figure 16: Memory-speedup diagram of the $767 \times 799 \times 551$ BRAIN+ data set. The times used for calculating the speedup factor are estimated times, not measured times.

only the estimated times, resp. speedup factors. On our 1 gigabyte machine, already the data set alone (644 MB) takes up most of the available main memory leaving little space (about 300 MB) for hybrid data structures, let alone a complete kd- or interval tree, which would have required 3 gigabytes. For this configuration our hybrid resulted in a speedup over enumeration by a factor of 45. This acceleration was achieved with a spatial partition tree in combination with the out-of-core interval tree method. No in-core kd- and interval trees were used. This result is better than what can be achieved using the plain out-of-core method (speedup 37) and also better than using only a spatial partition tree (speedup 6.1).

Figure 21 shows, how the available memory is split up between the four constiutuent methods of the hybrid, and how this division of space evolves as more memory is made available. Simularly, Figure 22 shows, how the spatial partitioning tree builds up and then shrinks as more memory gets allocated for the auxiliary data structures. For low memory, the space is used mostly for spatial subdivision and most blocks of cells are searched by the out-of-core method. As more memory becomes available, space is invested in nodes with kd-trees and interval trees, where the kd-tree part is smaller than the interval tree part. Eventually, interval trees conquer the entire volume shrinking the kd-tree part back to zero. Near the end,

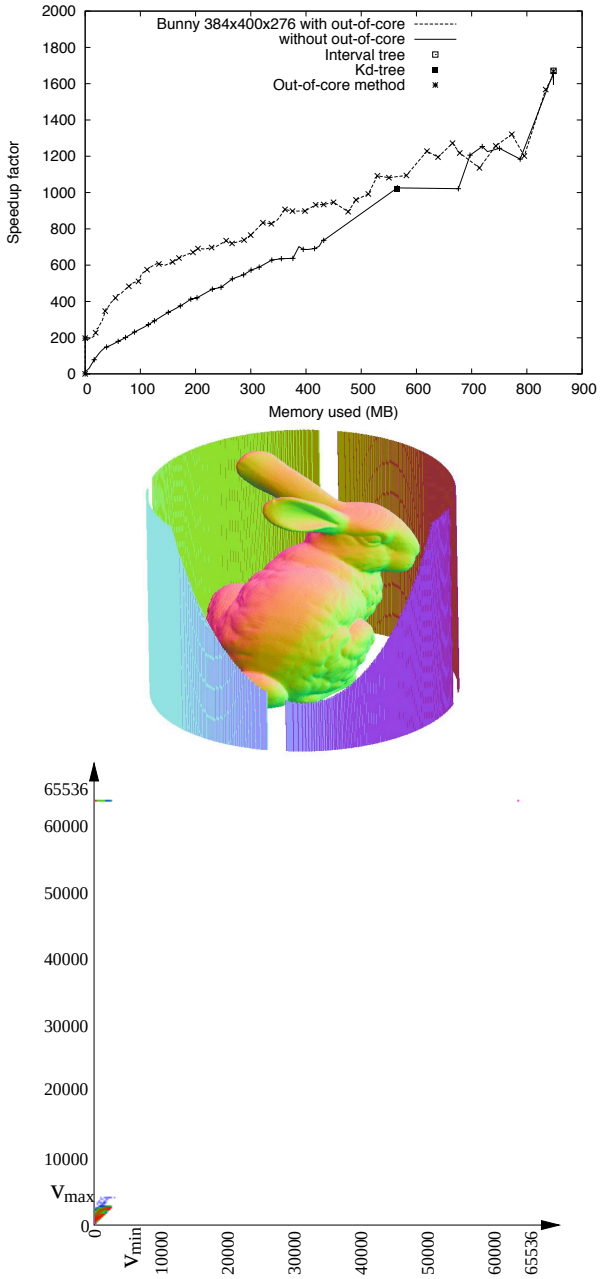
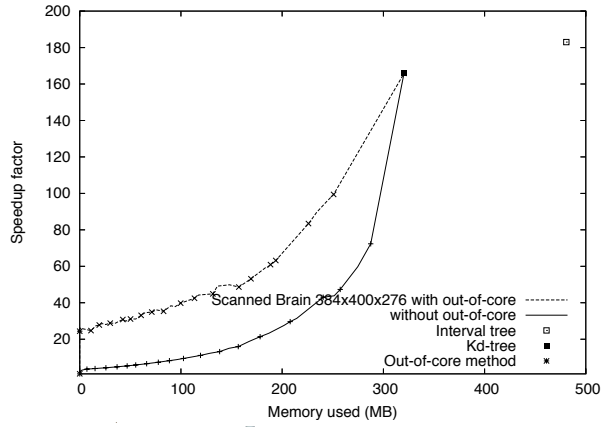


Figure 17: Isosurface, span space histogram and memory-speedup diagram of the $360 \times 512 \times 512$ terracotta BUNNY data set (180 MB). Average extraction time by enumeration is 4610 ms.



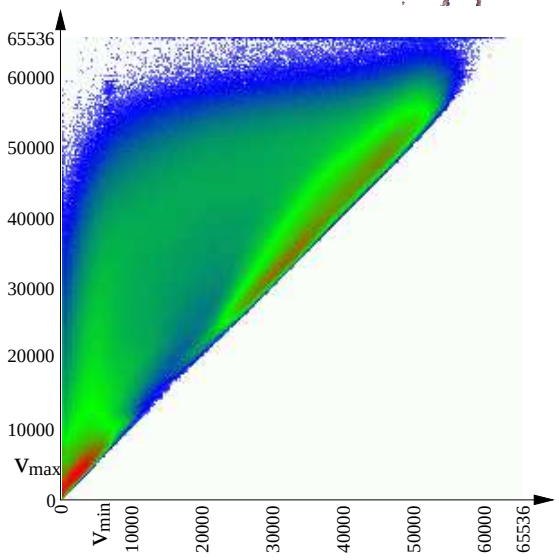
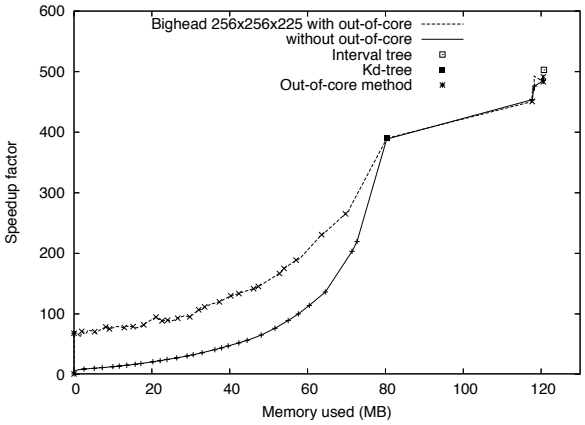
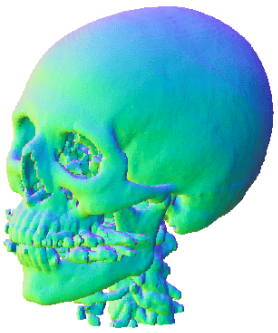


Figure 18: Isosurface, span space histogram and memory-speedup diagram of the $384 \times 400 \times 276$ BRAIN data set (81 MB). Average extraction time by enumeration is 1960 ms.





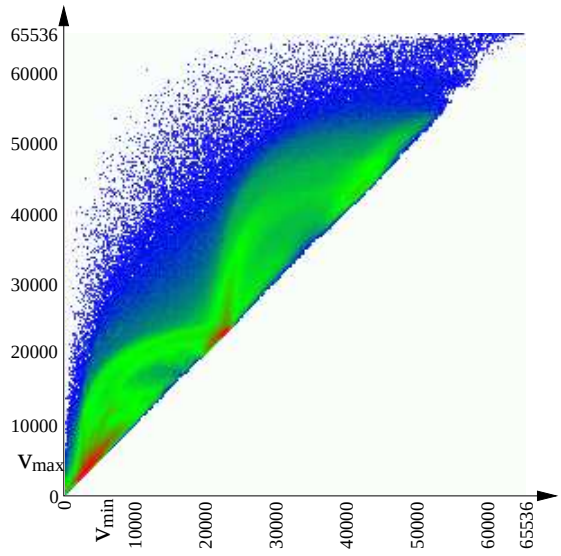
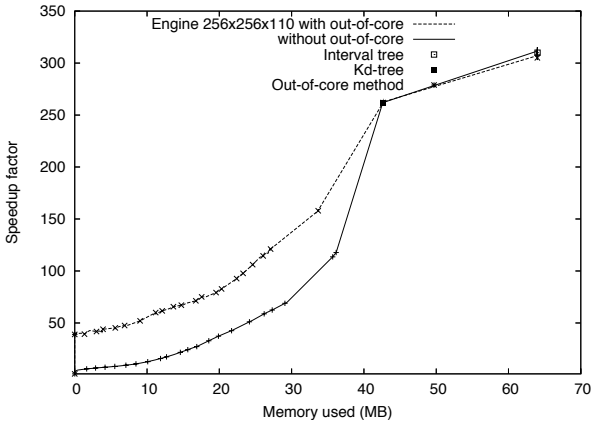
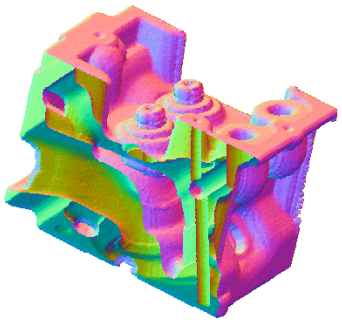


Figure 19: Isosurface, span space histogram and memory-speedup diagram of the $256 \times 256 \times 225$ BIGHEAD data set (28 MB). Average extraction time by enumeration is 635 ms.





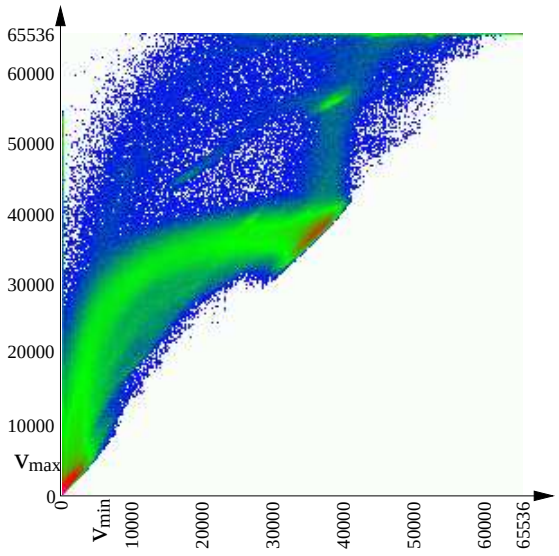


Figure 20: Isosurface, span space histogram and memory-speedup diagram of the $256 \times 256 \times 110$ ENGINE data set (14 MB). Average extraction time by enumeration is 318 ms.

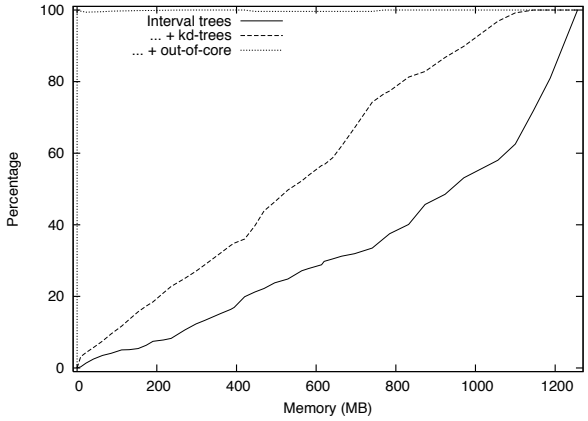


Figure 21: The percentage of cells examined with the interval tree, kd-tree and out-of-core method as a function of memory allocation for the XMAS data set.

the nodes of the partition tree are merged again, reducing the number of blocks. At the end, with 1256 MB, only one node remains, pointing at an interval tree.

For other data sets (not discussed here), we have observed similar results. In most cases, the fastest algorithm was an interval tree, specially in data sets with smoothly varying data values and small cell intervals.

6 Conclusion and future work

We have developed the theory and practise of optimal hybrid memory constrained cell extraction for isosurface generation from regularly structured volume data. The method is based on a spatial partition of the volume. In each block of the partition, one of several extraction methods is applied. In this paper we have considered enumeration (as in marching cubes), interval trees, kd-trees, and out-of-core interval trees. The method provides maximal active cell extraction speed by optimally chosing the spatial partition together with the methods to be assigned to the partition blocks, while observing a prescribed memory constraint for the required space of the auxiliary data structures. Our implementation and simulations demonstrate the effectiveness of the method. The hybrid provides faster isosurface extraction than prevous state-of-

the-art methods in cases where the best auxiliary data structures (e.g., kd- or interval trees) require more in-core memory than available.

A limitation of our method is that the Lagrangian optimization yields only optimal points that are on the lower convex hull of the set of operational points $(E[\mathcal{T}], \mathcal{M})$. This may lead to gaps in the curves as in Figure 20 from about 42 to 65 MB of memory. This is a general and well known fact about Lagrangian optimization, which often can be overcome by a more general dynamic optimization strategy [17]. For our case, however, this would amount to optimization by inefficient full search. We have devised simple heuristic strategies that yield suboptimal solutions in such gaps, see [21].

Other methods besides enumeration, interval trees, kd-trees and out-of-core interval trees, for example the ISSUE algorithm[19], or propagation algorithms[4, 9], may be integrated in the optimization framework, provided that a feasible model for the memory cost of the methods and an estimation of the extraction speed can be provided.

Our hybrid isosurface extraction method may be modified to apply also to irregularly structured volume data as opposed to the rectilinear case discussed in this paper. It is clear that the methods using kd- and interval trees can handle the irregular case as well without change. The binary space partition, however, does not directly apply to unstructured data. In this case one may still use a bsp-tree for the usual spatial partition, but the cells (i.e., tetrahedra or other polyeders) need to be assigned to the regions of the spatial partition. For example, one may assign a polyeder to a halfspace based on volume, or spatial extent. Then, when using the basic method by enumeration for cell extraction, one would store a list of indices that specifies the polyeders in the corresponding region of the spatial partition. Thus, the main difference to hybrid isosurface extraction for rectilinear data is that in the unstructured case the enumeration method is not memory free anymore.

Acknowledgment. We thank Frithjof Kruggel and the Max Planck Institute for Human Cognitive and Brain Sciences, Leipzig, for providing the MRI volume data set BRAIN and Jens Peer Kuska for its rendering in figure 18.

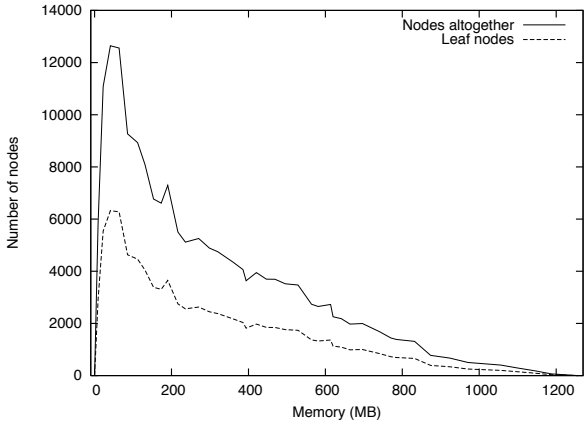


Figure 22: Nodes in optimal spatial partitioning tree as a function of memory for the XMAS data set.

References

[1] Sources of volume data sets:
XMAS: Technische Universität Wien,
http://www.cg.tuwien.ac.at/xmas/
BUNNY: Stanford Volume Data Archive,
http://graphics.stanford.edu/data/voldata
BIGHEAD, ENGINE: Chapel Hill Volume Rendering Test Data Set, Volume II,
ftp://www-graphics.stanford.edu/pub/volpack/data/
BRAIN: CT-scan, Max Planck Institute for Human Cognitive and Brain Sciences, Leipzig,
http://www.cns.mpg.de (not available online)
TESTSET: implicit algebraic Kummer surface with Perlin noise perturbartion,
http://www.inf.uni-konstanz.de/cgip/projects/.

[2] E. Allgower, K. Georg, *Numerical Continuation Methods*, Springer-Verlag, New York, 1990.

[3] C. Bajaj, V. Pascucci, D. Schikore, *Accelerated isocontouring of scalar fields*, Chapt. 3 in: Data Visualization Techniques, C. Bajaj, B. Krishnamurthy (eds.), John Wiley and Sons, 1999.

[4] C. Bajaj, V. Pascucci, D. Schikore, *Fast isocontouring for improved interactivity*, ACM Siggraph/IEEE Symposium on Volume Visualization (1996), pp. 39–46.

[5] C. Bajaj, V. Pascucci, D. Schikore, *Seed sets and search structures for optimal isocontour extraction*, Technical Report 99-35, Texas Institute of Computational and Applied Mathematics, University of Texas, Austin, 1999.

[6] U. D. Bordoloi, H.-W. Shen, *Space efficient fast isosurface extraction for large data sets*, Proceedings IEEE Visualization 2003. pp. 201–208.

[7] Y.-J. Chiang, C.T. Silva, *I/O optimal isosurface extraction*, Proceedings IEEE Visualization 1997, pp. 293–300.

[8] Y.-J. Chiang, C. T. Silva, W. J. Schroeder, *Interactive out-of-core isosurface extraction*, Proceedings IEEE Visualization 1998, pp. 167–174.

[9] P. Cignoni, P. Marino, C. Montani, E. Puppo, R. Scopigno, *Speeding up isosurface extraction using interval trees*, IEEE Trans. Visualization and Comp. Graphics 3 (1997) 158–170.

[10] H. Everett III, *Generalized Lagrange multiplier method for solving problems of optimum allocation of resources*, Operations Research 11 (1963) 399–417.

[11] T. Itoh, Y. Yamaguchi, K. Koyamada, *Fast isosurface generation using the volume thinning algorithm*, IEEE Trans. Visualization and Comp. Graphics 7,1 (2001) pp. 32–46.

[12] M. v. Kreveld, R. v. Oostrum, C. Bajaj, V. Pascucci, D. Schikore, *Contour trees and small seed sets for isosurface traversal*, Proceedings ACM Symp. on Comp. Geom. '97, Nice (France).

[13] D. T. Lee and C. K. Wong, *Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees,* Acta Informatica 9 (1977) 23–29.

[14] Y. Livnat, H.-W. Shen, C.R. Johnson, *A near optimal isosurface extraction algorithm using the span space*, IEEE Trans. Visualization and Comp. Graphics 2 (1996) 73–84.

[15] W.E. Lorensen, H.E. Cline, *Marching Cubes: a high resolution 3D surface construction algorithm*, ACM Computer Graphics 21 (1987) 163–196.

[16] P. Ning, J. Bloomenthal, *An evaluation of implicit surface tilers*, IEEE Comp. Graphics and Appl. 13,6 (1993) 33–41.

[17] A. Ortega, K. Ramchandran,, *Rate-distortion methods for image and video compression*, IEEE Signal Processing Magazine 15,6 (1998) 23–50.

[18] D. Saupe, J. Toelke, *Optimal memory constrained isosurface extraction*, Proceedings Vision Modeling and Visualization 2001.

[19] H.-W. Shen, C.D. Hansen, Y. Livnat, C.R. Johnson, *Isosurfacing in span space with utmost efficiency (ISSUE)*, Proc. IEEE Visualization 1996, pp. 287–294.

[20] H.-W. Shen, C.R. Johnson, *Sweeping simplices: a fast isosurface extraction algorithm for unstructured grids*, Proc. IEEE Visualization 1995, pp. 143–150.

[21] J. Toelke, *Der Conditioned Tree und andere Optimierungsverfahren der IsoflŁchen-Extraktion*, Dissertation, University Konstanz, 2006, to appear.

[22] J. Wilhelms, A. Van Gelder, *Octrees for faster isosurface generation*, ACM Trans. on Graphics 11 (1992) 201–227.