

Optimal memory constrained isosurface extraction

Dietmar Saupe, Jürgen Toelke

University of Leipzig, Computer Science Institute
Augustusplatz 10/11, 04109 Leipzig, Germany

Email: saupe, toelke@informatik.uni-leipzig.de

Abstract

Efficient isosurface extraction from large volume data sets requires special algorithms and data structures. Such algorithms typically either 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 data structures demand storage space that can be many times as large as the original volume data. In practice storage space is constrained and, therefore, new algorithms may be necessary that adapt the size of the data structures to the given limits. We present a hybrid algorithm which combines binary space partition (BSP) trees with fast search methods at some leaf nodes of the BSP-tree and memory-free linear search at the remaining leaf nodes. The method optimally trades off space for extraction speed.

1 Introduction

Isosurfaces provide a basic intuitive approach for the visualization of scalar volumetric data. 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 $f : D \rightarrow \mathbb{R}$ with $D \subset \mathbb{R}^3$ is given by the preimage $f^{-1}(t)$ for some isovalue $t \in \mathbb{R}$. In practice, f is given in terms of samples on a regular (rectilinear) or irregular grid. Medical 3d imaging modalities typically produce regular volume data. Irregular volume data often arise from finite element simulations. In this paper we restrict to the case of regular rectilinear volume data.

For simplicity let us label the grid points using integer coordinates (i, j, k) . 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 grid points as its corner grid points. For a cell $C_{i,j,k}$ with grid point (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. We may assume that the range of \hat{f} in a cell $C_{i,j,k}$ is given by the interval $[v_{\min}, v_{\max}]$ with $v_{\min} = \min V_{i,j,k}$, $v_{\max} = \max V_{i,j,k}$. 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 are the cells that intersect the isosurface.

Actually, it suffices to consider the half-open intervals $[v_{\min}, v_{\max})$, which forces an extraction algorithm to report only one intersecting cell in place of two in cases where the isosurface only touches a face of a cell. Moreover, this way a cell is listed only once in auxiliary data structures such as kd-trees or interval trees, see below.

To efficiently organize the extraction is the central problem addressed in some previous papers. For a survey see [1]. In this section we review the major methods and explain the cell extraction problem with a memory constraint that is considered and solved for the first time in this paper.

Method	Search trees (MB)	Total space (MB)	Time (ms)	Speed-up factor
Enumeration	0	80.86	9596	1
Octree	35.37	116.24	2641	3.63
kd tree	320.62	401.48	478	20.1
Interval tree	480.62	561.48	98	98.2

Table 1: Results for cell extraction for an MRI volume data set of size $384 \times 400 \times 276$ (Fig. 2). The timings are averaged and do not include time for cell output and for polygon computation.

The canonical method for cell extraction proceeds by *enumeration*, i.e., it sequentially checks all cells $C_{i,j,k}$ in the volume for the condition $t \in [v_{\min}, v_{\max}]$. This method was used in the “marching cubes” paper of Lorensen and Cline [7]. Besides storage for the volume data it does not require any additional memory while the cell extraction time is linear in the size of the volume data.

With a *hierarchical space subdivision* of the volume a considerable acceleration can be achieved at the cost of moderate additional CPU space. Wilhelms and Van Gelder proposed in [10] an octree subdivision for this purpose.

More substantial acceleration of the cell extraction can be provided by partitioning the volume into subsets consisting of cells $C_{i,j,k}$ with similar “spans” $[v_{\min}, v_{\max}]$. For example, these spans, represented as points $(v_{\min}, v_{\max}) \in \mathbb{R}^2$ in “span space”, can be organized in a kd-tree structure allowing rapid extraction of the non-empty cells, see Livnat et al [6], an approach called the NOISE-algorithm. The per cell space requirements for this data structure are given by space for the v_{\min} and v_{\max} values and a pointer to the cell in addition to storage for the tree structure. The same authors together with Hansen suggest in [9] 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.

As an alternative Cignoni et al [4] propose the use of interval trees. 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 [4]. With this result, the method is output sensitive optimal. In order to reduce the space requirements for the interval trees one may restrict to a subset of cells such that all edges in the volume grid are covered [4]. After cell extraction one propagates the isosurface on to the neighboring cells. Thus, a

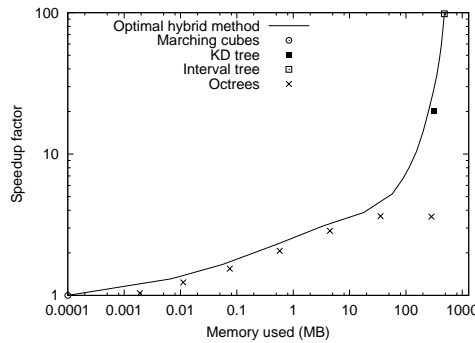


Figure 1: Comparison of methods for cell extraction for the MRI data set of Table 1. The solid line results from our adaptive optimal hybrid method.

tradeoff between space and time complexity is realized. However, the method is neither adaptive with respect to a given memory constraint nor optimally designed as the hybrid method in this paper.

Chiang and Silva present in [3] an interval tree method that works “out-of-core”, i.e., the data structure is stored externally on disk such that those parts that are needed for an extraction process can be efficiently read into main memory. Another method, proposed by Bajaj, Pascucci, and Schikore in [2], first searches a suitable subset of all cells for active cells from where the isosurface is generated by means of a fast propagation algorithm.

Let us consider the class of memory resident isosurface extraction methods. Table 1 shows an example for a data set of 16-bit values of resolution $384 \times 400 \times 276$ which amounts to about 81 megabytes (MB) of volume data. The three basic methods using an octree spatial subdivision, kd-tree and interval tree structures for the span space approach are explored next to the sequential search by enumeration without acceleration. With the search by enumeration the cell extraction time is 9.6 seconds averaged over random query values t (on a MIPS R12000 processor at 270 MHz). This rather long extraction time is sped up by an impressive factor of almost 100 using the interval tree method. However, the interval tree data structure requires 481 MB of memory for this achievement.¹ Now

¹The list structures in the interval tree require two pointers in the min- and max-lists (2×4 bytes) and the interval (2×2 bytes) for each cell. Overall, this amounts to 12 bytes for each cell in the interval tree. These are $383 \times 399 \times 275 - 27$, 845 cells (voxel values on the corners coincide in cells that are ignored).

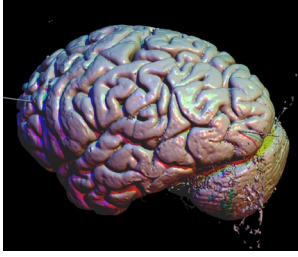


Figure 2: Ray traced isosurface from Table 1.

let us assume that only 200 MB memory for the auxiliary data structures is given. In this case the machine would be able to accommodate the octree method requiring 35 MB and providing an average extraction time of 2.6 seconds, while it would not be able to handle the kd-tree without swapping memory. 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 1. For the case of 200 MB of available memory the hybrid approach will result in an average extraction time of only 0.67 seconds.

In our hybrid approach we use a binary spatial partition (BSP). The leaves of the BSP tree correspond to regions that are searched using one method out of a pool of methods. Construction of the BSP tree and the selection of the search method is optimal in the sense that no other feasible configuration with the same or lower memory requirement can provide for a better acceleration.

2 Theory

Let us assume that we are given samples v_{ijk} of a scalar function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ at integer coordinates from which we can generate an interpolation $\hat{f} : \mathbb{D} \subset \mathbb{R}^3 \rightarrow \mathbb{R}$. Thus, $\hat{f}(i, j, k) = v_{ijk}$ for integers i, j, k in certain ranges. We define the cells C_{ijk} as explained in the last section and consider the set of eight corresponding voxel values $V_{ijk} = \{v_{i,j,k}, v_{i+1,j,k}, v_{i,j+1,k}, \dots, v_{i+1,j+1,k+1}\}$ and their convex hulls without the maximum point, $I_{ijk} = [\min V_{ijk}, \max V_{ijk})$. 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 the final step an isosurface renderer will compute and render $\hat{f}^{-1}(t) \cap C_{ijk}$ for all $(i, j, k) \in L$.

2.1 Memory constrained optimization

Assume that we are given a set of $m > 1$ search methods that solve the cell extraction problem. We consider *spatial partitions* P of the volume \mathbb{D} into blocks $B_k, k = 1, \dots, n$ each consisting of the union of a set of cells, $P = \{B_1, \dots, B_n\}, \mathbb{D} = \bigcup_{k=1}^n B_k$ such that each cell belongs to exactly one block B_k . In our hybrid scheme we assign to each block B_k one of the m search methods, denoted by the number $a_k \in \{1, \dots, m\}$. For a partition P with n sets, $|P| = n$, we thus have a *method assignment vector* $A = (a_1, \dots, a_n)$. Our *hybrid cell extraction* proceeds by sequentially processing the blocks B_1, \dots, 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 all blocks $B \subset \mathbb{D}$ and thresholds $t \in \mathbb{R}$. We use the notation $\mathcal{T}(B, a, t)$ where $a \in \{1, \dots, m\}$ denotes the search method. Likewise we denote by $\mathcal{M}(P), \mathcal{M}(B, a)$, 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 $P = \{B_1, \dots, B_n\}$ and method assignment vector $A = (a_1, \dots, a_n)$ is

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

A global *memory constraint* M can be written as $\mathcal{M}(P, A) \leq M$. Details for models of space and time for the methods are given in Section 3 and [8].

We also define a probability distribution for the thresholds $t \in \mathbb{R}$, because these isovalues are generally not known beforehand. E.g., 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 allows us to work with the expectation of the time requirement $\mathcal{T}(B, a, t)$ for

the cell extraction, denoted by $E[T(B, a, t)]$. The expectation of the total time required for the cell extraction in the entire volume \mathbb{D} using a partition P , $|P| = n$, with method assignment vector A is

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

Definition 1 (Memory constrained optimization)

For given volume data, m extraction methods, and a global memory constraint M define the configuration set

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

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

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

subject to the constraint $\mathcal{M}(P, A) \leq M$.

The problem as stated 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 discuss the case of a fixed partition and then solve the problem for a very large set of hierarchical partitions.

2.2 Optimization for fixed partitions

We assume for this subsection that the configuration space \mathcal{K} contains only one partition $P = \{B_1, \dots, B_n\}$. Thus, the task is to compute $A^* = \arg \min_{A \in \{1, \dots, m\}^{|P|}} E[T(P, A, t)]$ subject to the memory constraint $\mathcal{M}(P, A) \leq M$. Using a multiplier $\lambda \geq 0$ we convert the problem to an unconstrained discrete Lagrangian problem.

Definition 2 (Discrete Lagrangian problem)

Let P , $|P| = n$, be a partition of \mathbb{D} . For a given Lagrangian multiplier $\lambda \geq 0$ compute

$$A^* = \arg \min_{A \in \{1, \dots, m\}^n} E[T(P, A, t)] + \lambda \cdot \mathcal{M}(P, A). \tag{2}$$

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

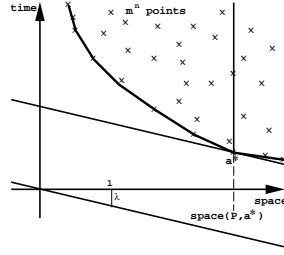


Figure 3: This space-time diagram for the unconstrained optimization contains a point $(\mathcal{M}(P, A), E[T(P, A, t)])$ for each of the m^n method assignment vectors A .

Proposition 1 Let $A^* \in \{1, \dots, m\}^n$ be a solution of the discrete Lagrange problem (2) for $\lambda \geq 0$. Then A^* solves the memory constrained optimization problem minimizing $E[T(P, A, t)]$ subject to $\mathcal{M}(P, A) \leq \mathcal{M}(P, A^*)$.

The proposition is a special case of a general theorem of Everett [5]. Figure 3 illustrates the proposition and suggests a proof of the statement.

To solve the constrained optimization problem for a given memory constraint M , one must solve (2) 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.

However, the solution of (2) by enumeration of all m^n points in the space/time diagram is not feasible because the complexity grows exponentially with the number n of blocks in the partition P . Fortunately, the exponential complexity can be reduced to a linear one by observing a separability property of the problem. The minimization in (2) is equivalent to minimizing the sum $\sum_{k=1}^n E[T(B_k, a_k, t)] + \lambda \cdot \mathcal{M}(B_k, a_k)$ whose terms are independent. Therefore we can treat the terms in the sum separately for $k = 1, \dots, n$, yielding solutions

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

2.3 Optimization for variable partitions

We now 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 ourselves 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 be subdivided in the same procedure and so on in a recursive manner. The partitioning of \mathbb{D} or of a macro block in \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 equally sized subsets. 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, which is a parameter of the hybrid cell extraction method.

Any 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 leaf nodes of the subtree rooted at ν . In particular, the root node corresponds to the entire volume \mathbb{D} , the other inner nodes to macro blocks, and the leaf nodes correspond to the blocks $B_k, k = 1, \dots, n$ of the hierarchical partition P .

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

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

where \mathcal{P} consists of all partitions P that correspond to pruned subtrees of P_{\max} .

Definition 4 (Global Lagrangian optimization)

Let P_{\max} be a maximal hierarchical partition of \mathbb{D} and let \mathcal{K} be the corresponding configuration set of hierarchical partitions as in Definition 3. The corresponding Lagrangian global cost function of $(P, A) \in \mathcal{K}$ is

$$J_\lambda(P, A) := E[\mathcal{T}(P, A, t)] + \lambda \cdot \mathcal{M}(P, A). \quad (3)$$

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

$$(P^*, A^*) = \arg \min_{(P, A) \in \mathcal{K}} J_\lambda(P, A). \quad (4)$$

Similar to Proposition 1 the solution (P^*, A^*) of the unconstrained global Lagrangian optimization

problem is also a solution to the constrained minimization of $E[\mathcal{T}(P, A, t)]$ satisfying $\mathcal{M}(P, A) \leq \mathcal{M}(P^*, A^*)$.

The cardinality of the set \mathcal{P} of partitions in \mathcal{K} is huge, exponential in the number of leaves of the maximal tree belonging to P_{\max} . Still, a solution is possible by observing a fundamental principle of optimality. To formulate this principle we first 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 $A_\nu \in \{1, \dots, m\}^{|P_\nu|}$ is 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

$$(P_\nu^*, A_\nu^*) = \arg \min_{P \in \mathcal{P}^{(\nu)}, A \in \{1, \dots, m\}^{|P|}} J_\lambda(P, A).$$

Proposition 2 (Principle of optimality) *Let $\lambda \geq 0$, P_{\max} the maximal hierarchical partition of \mathbb{D} , and (P^*, A^*) 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^* . Then the subtree P_ν at node ν in P^* together with the method assignment vector A_ν attached to the leaf nodes of P_ν provides the solution (P_ν^*, A_ν^*) of the local Lagrangian optimization problem relative to node ν .*

The proposition is due to the same separability property that we have explained in Subsection 2.2. In order for the global solution (P^*, A^*) to be optimal, every partition subtree of P^* 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.

The principle of optimality in Proposition 2 leads 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 is possible by using the solutions for the corresponding child nodes, requiring only a few comparisons of easily computed Lagrangian cost functions. Thus, local solutions are computed from the leaf nodes up 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_ν^* and the best method $A_\nu^* = (a)$ with

$$\begin{aligned} a &= \arg \min_{a=1, \dots, m} J_\lambda(P_\nu^*, (a)) \\ &= \arg \min_{a=1, \dots, m} E[T(B_\nu, a, t)] + \lambda \cdot \mathcal{M}(B_\nu, a) \end{aligned}$$

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

$$\begin{aligned} \min_{P_\nu, A_\nu} J_\lambda(P_\nu, A_\nu) &= \min \left\{ \min_{a=1, \dots, m} J_\lambda(P_\nu^0, (a)), \right. \\ &\left. \Delta_{E[T]}(\nu) + \lambda \Delta_{\mathcal{M}} + \sum_{\substack{\text{children} \\ \mu \text{ of } \nu}} \min_{P_\mu, A_\mu} J_\lambda(P_\mu, A_\mu) \right\} \end{aligned} \quad (5)$$

The first term takes care of the case in which the volume belonging to node ν is not further partitioned and searched using only one 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 $\Delta_{\mathcal{T}}(\nu) + \lambda \Delta_{\mathcal{M}}$, 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 $\Delta_{\mathcal{T}}(\nu)$ reflects the expected time required checking the node ν during cell extraction. 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.

2.4 Summary of the optimization method

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 large set of parameters $\lambda \geq 0$. Then the optimal solutions together with the corresponding space requirements are available for multiple cell extraction passes obeying an arbitrary memory constraint for the required auxiliary data structures. The preprocessing needs to be carried out only once. The result 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. Choose a sequence $(\lambda_i)_i$ with $0 < \lambda_1 < \lambda_2 < \dots < \lambda_l$ and a resolution parameter $\delta > 0$.
3. Build full partitioning tree P_{\max} using a maximal block size (e.g., $2 \times 2 \times 2$) for the termination criterion at the leaf nodes.
4. For all nodes ν and methods $a = 1, \dots, m$ estimate the required space $\mathcal{M}(B_\nu, a)$ and the expected extraction time $E[T(B_\nu, a, t)]$.
5. For each λ_i do the Lagrangian optimization obtaining optimal partition P_i^* , assignment vector A_i^* , and required space $\mathcal{M}(P_i^*, A_i^*)$.
6. Insert more λ -values and go to Step 5 to ensure that $\mathcal{M}(P_{i+1}^*, A_{i+1}^*) / \mathcal{M}(P_i^*, A_i^*) \leq 1 + \delta$.

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 $P_{i_M}^*$ and method assignment vector $A_{i_M}^*$ from the preprocessing, i.e., compute $i_M = \min\{i = 1, \dots, l \mid \mathcal{M}(P_i^*, A_i^*) \leq M\}$.
4. Build auxiliary data structures as given by $P_{i_M}^*$ and $A_{i_M}^*$.
5. User input: threshold t .
6. Extract the cells using the methods and data structures generated in Step 4.
7. Produce and display the polygonal approximation of the isosurface in each extracted cell.
8. If desired go to Step 5.

3 Implementation and models for space and time

Optimizing the expected cell extraction time requires a model of the space and time needed for the basic cell extraction methods. Here we choose

just two extraction methods for this analysis (thus, $m = 2$), namely the approach by enumeration ($a = 1$) and the method based on interval trees ($a = 2$). This choice is motivated by the fact that these methods provide the two extreme points in the space-time diagram, see Figure 1. As a result of the optimization we should obtain a scalable method that ranges from zero space requirement and high extraction times (by enumeration) to very large space requirements for the fastest possible extraction (by interval trees). The space requirements are zero for $a = 1$ and for $a = 2$ can be readily measured in the implementation without building the min- and max-lists for the interval trees. It is more difficult to estimate the time requirements.

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

We now consider the extraction method by enumeration. In the straightforward implementation one checks all cells in B_ν independently, requiring the same constant time α_0 . Thus, the expected extraction time at node ν using the extraction method by enumeration ($a = 1$) can be modelled by $E[T(B_\nu, 1, t)] = Pr(I_\nu) \cdot \alpha_0 |B_\nu|$. A slightly faster way to extract cells by enumeration checks a cell using the minimum and maximum values of 4 of its 8 voxel values which are already known from the previously processed cell. The model for expected extraction time can be adapted to this case requiring two parameters, β_0, β_1 , $E[T(B_\nu, 1, t)] = Pr(I_\nu) \cdot (i_1 - i_0)(j_1 - j_0)[(k_1 - k_0)\beta_0 + \beta_1]$.

A simple model for the expected cell extraction time for the method using interval trees ($a = 2$) 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, $\sum_{C_{ijk} \subset B_\nu} Pr(I_{ijk})$. However, we have implemented a more accurate model that takes into account the processing of the interval tree besides the scan of the min- and max-lists of the tree [8].

We model the expected time for checking an inner node in equation (5), $\Delta_{E[T]}(\nu)$, as $Pr(I_\nu) \cdot \alpha_3$.

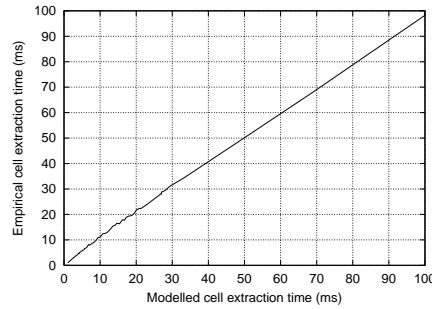


Figure 4: Modelled expected cell extraction time versus actual time averaged over different isovalues.

The model for expected extraction time contains parameters, that depend only on the machine used and can be determined empirically for some sample data set. We use several partitioning trees P_i and method assignment vectors A_i . It does not matter whether they are optimal. In each case i the model predicts a run time \hat{T}_i as a linear combination of the parameters. Then for each i the actual extraction times are measured and averaged for a set of query values obeying the underlying probability distribution, yielding empirical expected times T_i . Now the parameters can be determined using least squares minimization to fit the model to the data. E.g., the results using a Silicon Graphics Origin 200 processor at 270 MHz yielded values of 24.2 and 92.2 nano seconds for the parameters α_0 and α_3 , respectively. The time for output of the cells is not considered because overall it is constant and irrelevant for the optimization. See Figure 4 for results.

We conclude this section with remarks regarding our implementation. The required space and expected extraction times are computed using a recursive depth-first tree traversal of the maximal partitioning tree. In the entire process no min- and max-lists for interval trees need to be built. It suffices to know the structure of the interval 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 . These bottom-up Lagrangian optimizations are also computed using the recursive depth-first tree traversal algorithm. The resulting optimal hierarchical partitions P_i^* are stored along with the assignment vectors A_i^* .

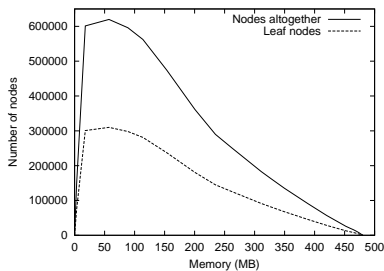


Figure 5: Nodes in optimal spatial partitioning tree as a function of memory allocation.

4 Results and future work

We have applied our optimal hybrid method to several data sets, but give results here only for the MRI data set from Section 1, compare Figure 1. Figure 5 shows how the spatial partitioning tree builds up and then shrinks as more memory gets allocated for the auxiliary data structures. The first 18 MB are used entirely for spatial subdivision up to about 300,000 blocks all of which are searched by enumeration. From then on additional space is invested almost exclusively in nodes with interval trees and eventually such nodes are merged reducing the number of blocks. At the end, with 480 MB, only one node remains, i.e., the root node which belongs to an interval tree encompassing the entire volume. For other data sets we have observed similar results.

Our method is practical. The preprocessing time in Algorithm 1 is dominated by Step 4 in which the time and space requirements for all methods and all nodes of the maximal partitioning tree are computed. On our machine (Silicon Graphics Origin 200 R12000 processor at 270 MHz) these computations take about 360 seconds for the 81 MB MRI data set. We store the results of the preprocessing in a disk file which is accessed by Algorithm 2.

For future work we remark that for the cell extraction we may join all the leaf nodes of the BSP tree that are searched with the interval tree method. This way only one interval tree needs to be built reducing space requirements. Another improvement providing an accelerated cell extraction can be obtained eliminating the partition tree altogether keeping only the leaves ν , which must be traversed for each cell extraction. This traversal can be efficiently implemented using another interval tree, this time

for the intervals I_ν , i.e., the ranges of values in the volumes belonging to the leaf nodes.

Other methods besides enumeration and interval trees such as the one based on kd-trees [6], the ISSUE algorithm [9], propagation algorithms [2, 4] and even external out-of-core algorithms [3] can be integrated in our optimization framework as well.

Acknowledgment. We thank F. Kruggel for the MRI data and J. P. Kuska for Figure 2.

References

- [1] C. Bajaj, V. Pascucci, D. Schikore, *Accelerated isocontouring of scalar fields*, Data Visualization Techniques, C. Bajaj, B. Krishnamurthy (eds.), John Wiley and Sons, 1999.
- [2] C. Bajaj, V. Pascucci, D. Schikore, *Fast isocontouring for improved interactivity*, ACM Siggraph/IEEE Symposium on Volume Visualization (1996), pp. 39–46.
- [3] Y.-J. Chiang, C.T. Silva, *I/O optimal isosurface extraction*, Proceedings IEEE Visualization 1997, pp. 293–300.
- [4] P. Cignoni, P. Marino, C. Montani, E. Puppo, R. Scopigno, *Speeding up isosurface extraction using interval trees*, IEEE Trans. Visualization Comp. Graphics 3 (1997) 158–170.
- [5] H. Everett III, *Generalized Lagrange multiplier method for solving problems of optimum allocation of resources*, Operations Research 11 (1963) 399–417.
- [6] Y. Livnat, H.-W. Shen, C.R. Johnson, *A near optimal isosurface extraction algorithm using the span space*, IEEE Trans. Visualization Comp. Graphics 2 (1996) 73–84.
- [7] W.E. Lorensen, H.E. Cline, *Marching Cubes: a high resolution 3d surface construction algorithm*, ACM Comp. Graphics 21 (1987) 163–196.
- [8] D. Saupe, J. Toelke, *Optimal memory constrained isosurface extraction*, Tech. report, Institut für Informatik, Univ. Leipzig, 2001.
- [9] 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.
- [10] J. Wilhelms, A. Van Gelder, *Octrees for faster isosurface generation*, ACM Trans. on Graphics 11 (1992) 201–227.