History Trees as Descriptors of Macromolecular Structures

Deniz Sarioz[1], T. Yung Kong[2], and Gabor T. Herman[1]

[1] Ph.D. Program in Computer Science, The Graduate School and University Center
City University of New York, New York, NY 10016, USA
sarioz@acm.org gaborherman@yahoo.com
[2] Department of Computer Science, Queens College,
City University of New York, Flushing, NY 11367, USA
ykong@cs.qc.edu

Abstract. High-level structural information about macromolecules is now being organized into databases. One of the common ways of storing information in such databases is in the form of three-dimensional (3D) electron microscopic (EM) maps, which are 3D arrays of real numbers obtained by a reconstruction algorithm from EM projection data. We propose and demonstrate a method of automatically constructing, from any 3D EM map, a topological descriptor (which we call a history tree) that is amenable to automatic comparison.

Key words: Discrete shape representation; digital topology; macromolecular structures; volume images.

1 Introduction

High-level structural information about macromolecules is now being organized into databases such as the Quaternary Protein Structure (QPS) and the Electron Microscopy Data Base (EMDB), both at the European Bioinformatics Institute (EBI). Initiatives in the EM field are also starting in the US, nucleated around the Research Collaboratory for Structural Bioinformatics (RCSB) that is responsible for the database called the Protein Data Bank (PDB).

These databases include reconstructions from EM data, i.e., 3D arrays of real numbers that are voxelizations of macromolecular structures. Suppose that a biology researcher has obtained from EM projections a new reconstruction of the structure of a macromolecule, and would like to see if a database contains a similar object. The very large size of these 3D arrays, the arbitrary position and orientation of the molecule in the array and the possibility of non-linear stretching of the range make standard methods of comparison infeasible. Hence, there is a need for exploring and analyzing topological and geometrical features of the contents of such large aggregates in a systematic, quantitative and automatic manner to mine the information contained in these databases. In the following we propose a method of automatically producing topological descriptors of 3D EM arrays, and demonstrate it on arrays that we acquired from EBI. We believe that

comparison of these descriptors using appropriately defined similarity measures will be useful in the identification and classification of macromolecules.

2 Mathematical Preliminaries

2.1 Foreground Components in 3D Images; f-Ancestors

An EM reconstruction is typically represented as a 3D image — i.e., a real-valued mapping f defined on the voxel set X of a box-shaped region. For every real number t, we define the *foreground voxel set* of f at t as: $\mathcal{F}_f(t) = \{x \in X \mid f(x) \geq t\}$. Note that $\mathcal{F}_f$ shrinks monotonically: if $t_1 < t_2$, then $\mathcal{F}_f(t_1) \supseteq \mathcal{F}_f(t_2)$. We partition each foreground voxel set $\mathcal{F}_f(t)$ into connected components based on 6-connectivity [1]. [Two voxels are 6-*adjacent* if they share a face. A 6-*path* in U is a sequence of voxels in U in which every consecutive pair of voxels are 6-adjacent. A set U of voxels is 6-*connected* if for every pair of voxels in U there is a 6-path in U that begins at one voxel of the pair and ends at the other. A 6-*component* of U is a maximal non-empty 6-connected subset of U.]

Let $C_f(t)$ denote the collection of 6-components of $\mathcal{F}_f(t)$ and let $\mathcal{C}_f = \bigcup_{t \in \mathbb{R}} C_f(t)$. If $D_1, D_2 \in \mathcal{C}_f$, then we say D_1 is an *f-ancestor* of D_2 if there exist $t_1 \leq t_2$ such that $D_1 \in C_f(t_1)$, $D_2 \in C_f(t_2)$ and $D_1 \supseteq D_2$.

2.2 Foreground History Trees

We define the *ancestor* and *descendant* relations on the vertices of a rooted tree recursively, as follows (cf. [2, p. 93]): If v_1 and v_2 are vertices of a rooted tree, then v_1 is an ancestor of v_2 (and v_2 is a descendant of v_1) if $v_1 = v_2$ or v_1 is an ancestor of the parent of v_2.

A *foreground history tree* (FHT) for an image f is a rooted tree T in which each vertex v is associated with a real number $L_T(v)$, called its *level*, and a set $D_T(v) \in C_f(L_T(v))$, and in which the following conditions are satisfied:

1. If $L_T(v_1) = L_T(v_2)$ and $D_T(v_1) = D_T(v_2)$, then $v_1 = v_2$.
2. A vertex v_1 is an ancestor in T of a vertex v_2 if, and only if, $L_T(v_1) \leq L_T(v_2)$ and $D_T(v_1)$ is an f-ancestor of $D_T(v_2)$.

We will usually omit the subscript T from L_T and D_T unless it is needed to distinguish the L and D functions of different FHTs.

Note that if a vertex v_1 is the parent of a vertex v_2 in an FHT, then we have that $L(v_1) < L(v_2)$. Indeed, v_1 is an ancestor of v_2, and so $L(v_1) \leq L(v_2)$ and $D(v_1)$ is an f-ancestor of $D(v_2)$. Suppose $L(v_1) = L(v_2)$. Then $D(v_1)$ and $D(v_2)$ would both be 6-components of $\mathcal{F}_f(L(v_1)) = \mathcal{F}_f(L(v_2))$, and since $D(v_1) \supseteq D(v_2)$ (because $D(v_1)$ is an f-ancestor of $D(v_2)$) this would imply $D(v_1) = D(v_2)$. But then $v_1 = v_2$ (as $L(v_1) = L(v_2)$) contrary to the fact that v_1 is the parent of v_2.

FHTs are related to contour trees [3]; but contour trees are not necessarily rooted, whereas FHTs are. It is our hypothesis that FHTs can be made to reflect certain essential properties of macromolecules, and so provide a suitable basis for assessing the similarity of macromolecules.

3 Methods and Results

3.1 Obtaining a Foreground History Tree

Let $f : X \to \mathbb{R}$ be a 3D image, and let $\tau_1 > \tau_2 > \ldots > \tau_k$ be a strictly decreasing finite sequence of real numbers such that $f_{\max} \geq \tau_1$ and $f_{\min} \geq \tau_k$, where $f_{\max}$ and $f_{\min}$ are the maximum and the minimum values attained by f on the set of voxels X. [Thus $\emptyset \subsetneq \mathcal{F}_f(\tau_1) \subseteq \mathcal{F}_f(\tau_2) \subseteq \ldots \subseteq \mathcal{F}_f(\tau_k) = X$, and $|C_f(\tau_k)| = 1$.]

We now describe an algorithm that constructs an FHT T for f which satisfies the condition $\{L(v) \mid v \in \text{Vertices}(T)\} = \{\tau_i \mid 1 \leq i \leq k\}$, and which has as many vertices as is possible for such an FHT. For convenience in describing the algorithm, we define τ_0 to be an arbitrary but fixed number that exceeds $f_{\max}$, so that $\mathcal{F}_f(\tau_0) = \emptyset$ and hence $C_f(\tau_0) = \emptyset$.

The algorithm is based on a data structure which represents a collection of pairwise disjoint nonempty sets of voxels. Operations MAKE_SET(x) and UNITE_SETS(x_1, x_2) are used to change the represented collection of sets. These are defined below the next paragraph, whose aim is to provide the reader with an initial understanding of how the algorithm alters the data structure and, at the same time, builds up the FHT.

The algorithm has a main loop whose body is executed k times. At the end of the ith iteration of the loop, the data structure represents the collection of sets $C_f(\tau_i)$ and the FHT has been built (from its leaves towards its root) up to the level τ_i; i.e., all vertices v in the eventual FHT for which $L(v) \geq \tau_i$ have been created, and both $L(v)$ and $D(v) \in C_f(L(v))$ have been determined for these vertices, as well as the parent-child relationships among them. Clearly, at the end of the kth iteration, we have produced the whole FHT.

When the data structure represents a collection $\mathcal{S}$ (of disjoint sets of voxels), and x is any voxel such that $x \notin \bigcup \mathcal{S}$, a call of MAKE_SET($x$) adds the singleton set $\{x\}$ to the represented collection of sets: It changes the data structure from a representation of $\mathcal{S}$ to a representation of $\mathcal{S} \cup \{\{x\}\}$.

The algorithm calls UNITE_SETS either to replace two existing sets of voxels in the represented collection of sets with the union of those two sets, or to insert a new voxel, which does not yet belong to any set in the represented collection, into one of the sets in the collection. The effect of calling UNITE_SETS can be more precisely described as follows. Let x_1 and x_2 be voxels, and let $\mathcal{S}$ be the collection of sets that is represented by the data structure when UNITE_SETS(x_1, x_2) is called. For $i = 1, 2$, let $S_i = \{x_i\}$ if $x_i \notin \bigcup \mathcal{S}$, and let S_i be the member of $\mathcal{S}$ that contains x_i if $x_i \in \bigcup \mathcal{S}$. Then the call UNITE_SETS(x_1, x_2) changes the data structure from a representation of $\mathcal{S}$ to a representation of the collection $(\mathcal{S} \setminus \{S_1, S_2\}) \cup \{S_1 \cup S_2\}$.

The data structure is in fact a forest of rooted trees of nodes, together with a list of the root nodes of all the trees in the forest. Our data structure is an example of a *disjoint-set forest* (DSF) [2, pp. 446–450]. Trees and nodes of our DSF will be called DSFtrees and DSFnodes. The list of root nodes is given by the variable CURRENT_DSF_ROOTS, which is updated by UNITE_SETS and MAKE_SET.

There is a 1-to-1 correspondence between the collection of all DSFtrees of the DSF and the collection of disjoint sets of voxels that is represented by the DSF. Each voxel x has a field x.DSFnode that can either be a DSFnode or be NIL. A voxel x belongs to the set of voxels that corresponds to a DSFtree $\mathcal{T}$ if, and only if, x.DSFnode is a DSFnode in $\mathcal{T}$. Thus x.DSFnode is NIL if, and only if, x does not belong to any set in the collection of sets that is represented by the DSF. If x.DSFnode is NIL, then a call of MAKE_SET(x) will create a new DSFtree that consists of one new DSFnode and will set x.DSFnode to be that new DSFnode. This is the only way in which the algorithm creates DSFnodes.

The role of a DSFnode in our algorithm is analogous to the role of a label in standard connected component labeling algorithms for binary images (see, e.g., [4, pages 347–349]). DSFnodes that belong to the same DSFtree correspond to "equivalent" labels (i.e., labels that represent the same component).

The algorithm uses a function DSF_ROOT(), which is such that if ν is any DSFnode then DSF_ROOT(ν) returns the root of the DSFtree which contains ν. Thus two voxels x and y belong to the same member of the collection of sets that is represented by the DSF if, and only if, x.DSFnode $\neq$ NIL, y.DSFnode $\neq$ NIL and DSF_ROOT(x.DSFnode) = DSF_ROOT(y.DSFnode).

The algorithm starts by scanning the voxels in X and assigning each of them to one of k voxel "bins" $B[1], B[2], \ldots, B[k]$ as follows: A voxel $x \in X$ is placed in the bin $B[j]$, where j is the integer such that $\tau_{j-1} > f(x) \geq \tau_j$. [Thus the set of voxels that are placed in each bin $B[i]$ is just $\mathcal{F}_f(\tau_i) \setminus \mathcal{F}_f(\tau_{i-1})$.] The DSF is initialized to be empty, so x.DSFnode is NIL for every voxel x in X, and CURRENT_DSF_ROOTS is an empty list. [It follows that, for all voxels x at all times during the execution of the algorithm, x.DSFnode is NIL if, and only if, x has never been an argument of a call of MAKE_SET() or UNITE_SETS().]

After this initialization, the rest of the algorithm is stated by the pseudocode below. The loop on lines 4–7 transforms the DSF from a representation of $C_f(\tau_{i-1})$ to a representation of $C_f(\tau_i)$. [Note that x.DSFnode must be NIL on entry to the inner loop on lines 5–6, because the voxel x will not have been an argument of any earlier call of MAKE_SET() or UNITE_SETS(). So MAKE_SET(x) will be called on line 7 if, and only if, y.DSFnode = NIL for every 6-neighbor y of x.] Lines 8–11 create the vertices of the FHT that have level τ_i, and assign the children to the newly created vertices.

```
1. for i ← 1 to k do
2.     foreach ν ∈ CURRENT_DSF_ROOTS do ν.oldVertex ← ν.vertex;
3.     previousDSFroots ← a copy of the list CURRENT_DSF_ROOTS;
4.     foreach x ∈ B[i] do
5.         foreach 6-neighbor y of x in X do
6.             if y.DSFnode ≠ NIL then UNITE_SETS(x, y);
7.         if x.DSFnode = NIL then MAKE_SET(x);
8.     foreach ν ∈ CURRENT_DSF_ROOTS do
9.         ν.vertex ← a new FHT vertex v with L(v) = τ_i and D(v) = ν;
10.    foreach ν ∈ previousDSFroots do
11.        Make (DSF_ROOT(ν)).vertex the parent of ν.oldVertex;
```

The UNITE_SETS(x, y) operation and the DSF_ROOT(ν) function are implemented using union-by-rank with path-compression [2, p. 447]. The running time of a sequence of m calls of MAKE_SET, UNITE_SETS and DSF_ROOT is $O(m\beta(m))$, where $\beta(m)$ is an *extremely* slowly growing function of m [2, p. 449]. [$\beta(m) = \alpha(m, m)$, where α is an inverse of Ackermann's function.] In fact $\beta(m) = 3$ or 4 for all values of $m \geq 8$ that might occur in any conceivable application. In our applications the number k of levels or bins is very much smaller than the number $|X|$ of voxels in the domain X of the image f, and $\sum_{i=1}^{k} |C_f(\tau_i)|$ is also smaller than $|X|$. In this context the time complexity of the algorithm is $O(|X| \log k + |X|\beta(|X|))$, where the term $|X| \log k$ corresponds to the time complexity of initializing the k bins, and the factor $\log k$ assumes the use of binary search to find the appropriate bin for each voxel. As $\beta(|X|) = 3$ or 4 for all images of practical interest, the time-complexity is "essentially" $O(|X| \log k)$. If the levels τ_i are regularly spaced, then the bin for each voxel can be found in $O(1)$ time and the time-complexity of the algorithm is essentially $O(|X|)$.

3.2 Preliminary Results and the Need for Simplification

We applied our method to several macromolecules, and here we present its application to a reconstruction of the e. coli 70s ribosome [5], and a helical reconstruction of drosophila kinesin dimer AMP-PNP state [6]. Figs. 1 and 2 show surface visualizations and cross-sections of these specimens. Figs. 3(a) and 4(a) show associated FHTs. We used the drawgram program in the package PHYLIP [7] to generate these tree images. Each vertex that has more than one child is represented by a horizontal segment. The presence of a downward segment from a horizontal segment a to a horizontal segment or endpoint d indicates that the vertex represented by d is a descendant of the vertex represented by a, and the length of the downward segment is proportional to the difference between the levels of those vertices. In each case, we eliminated the vertical segment from the root of the tree.

These FHTs are given by the algorithm of Sect. 3.1 with $k = 128$ and, for $1 \leq i \leq 128$, $\tau_i = f_{\max} - i\Delta$, where $\Delta = (f_{\max} - f_{\min})/128$.

While they appear to be different, the trees of Figs. 3(a) and 4(a) are so cluttered that one cannot be sure whether or not the difference is just an artifact of the display. We need to construct simpler FHTs that better reveal the structural essence of the molecules.

3.3 Pruning FHTs by Component Size

Pruning an FHT by component size removes all vertices that correspond to components containing fewer than a certain number of voxels. This transforms an FHT T for an image f to an FHT T' for f such that Vertices$(T') = \{v \in \text{Vertices}(T) \mid |D_T(v)| \geq \delta\}$, in which the functions $L_{T'}$ and $D_{T'}$ are the restrictions to Vertices(T') of L_T and D_T. Here δ is a positive integer parameter that represents the minimum allowed component size. [This operation cannot

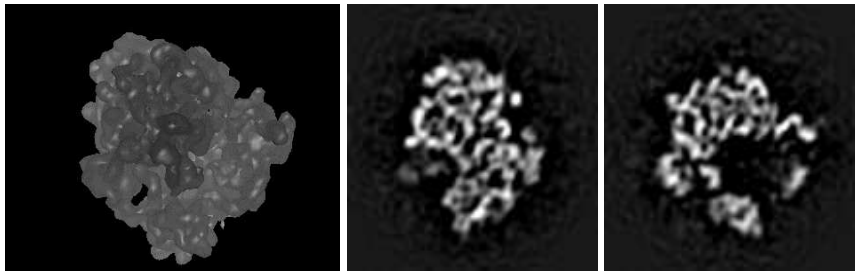


Fig. 1. EMD-1006: E.coli 70s ribosome / ribosome-bound termination factor RF2, surface visualization and cross-sections (51 and 80 of 130).

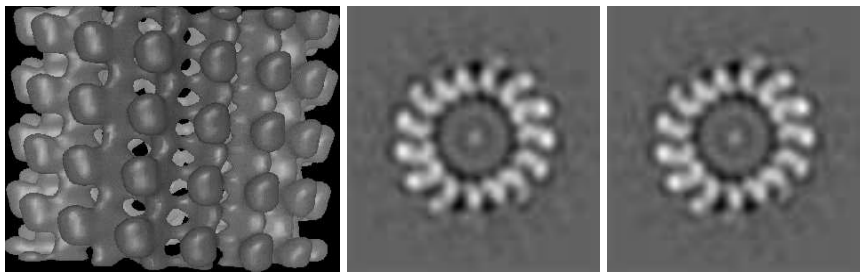
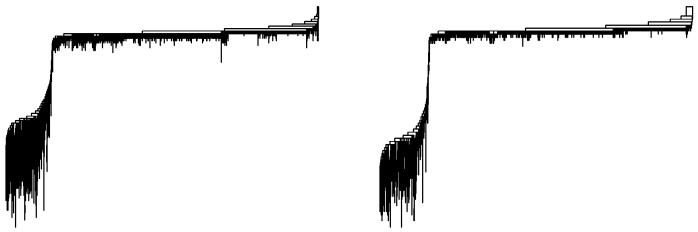
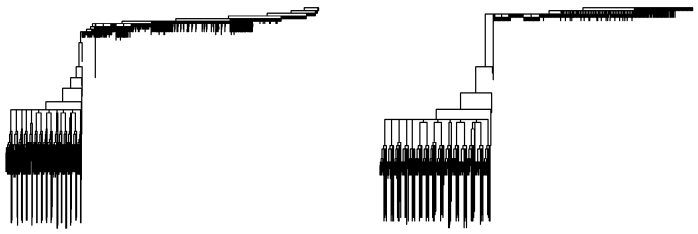


Fig. 2. EMD-1032: Drosophila kinesin dimer AMP-PNP state, surface visualization and cross-sections (47 and 63 of 100).



(a) (b)

Fig. 3. FHTs based on EMD-1006: (a) unpruned, (b) pruned by minimum component size of 25.



(a) (b)

Fig. 4. FHTs based on EMD-1032: (a) unpruned, (b) pruned by minimum component size of 20.

disconnect the tree, for if a vertex u of an FHT is the parent of a vertex v, then $|D(v)| \geq \delta$ implies $|D(u)| \geq \delta$.]

It is easy to incorporate pruning by component size into the algorithm of Sect. 3.1 for producing an FHT. The pseudocode need not be changed. It suffices to store the size of the set of voxels that is represented by each DSFtree in a field of its root DSFnode, and include in the list CURRENT_DSF_ROOTS just those root DSFnodes whose size fields are greater than or equal to δ. The algorithm will then construct the tree that would be obtained if we pruned the FHT that is produced by the original version of the algorithm.

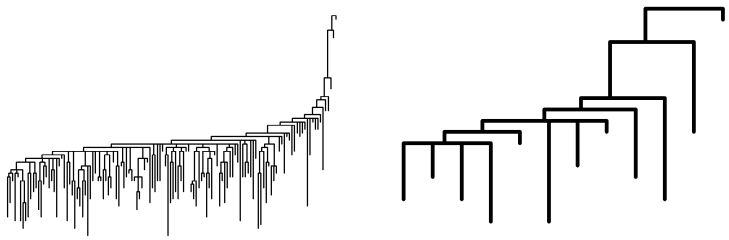
Figs. 3(b) and 4(b) show pruned FHTs of the images of Figs. 1 and 2.

3.4 Pruning FHTs by Subtree Height

The FHTs that are constructed as described above tend to have "comb-like" structures: There are many leaves near the root. [See the right side of each tree in Figs. 3 and 4.] These leaves correspond to components of the foreground voxel set at very low gray values, and seem mostly to represent components of the ice in which the specimen under study is embedded.

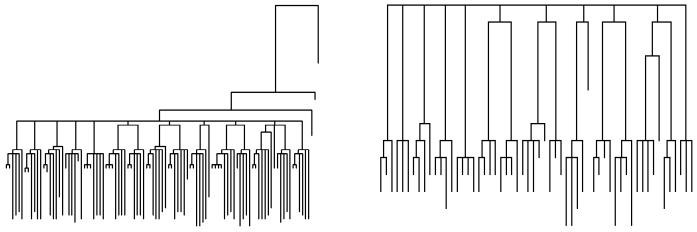
Pruning by subtree height is a second method of pruning that can be applied to FHTs. It eliminates the above-mentioned leaves and some other artifacts that are likely to be due to noise.

Pruning by subtree height h transforms an FHT T for an image f to an FHT T' for f such that $\text{Vertices}(T') = \{v \in \text{Vertices}(T) \mid \text{subtree-height}_T(v) \geq h\}$, in which the functions $L_{T'}$ and $D_{T'}$ are the restrictions to $\text{Vertices}(T')$ of L_T and D_T. Here $\text{subtree-height}_T(v) = \max\{L(w) - L(v) \mid w \text{ is a descendant of } v \text{ in } T\}$. Figs. 5 and 6 show the effects of this operation on the FHTs of Figs. 3(b) and 4(b), for two different values of h.



(a) (b)

Fig. 5. FHTs based on EMD-1006: minimum component size of 25, pruned by subtree height parameter (a) $h = 4\Delta$, (b) $h = 20\Delta$, where $\Delta = (f_{\max} - f_{\min})/128$.



(a) (b)

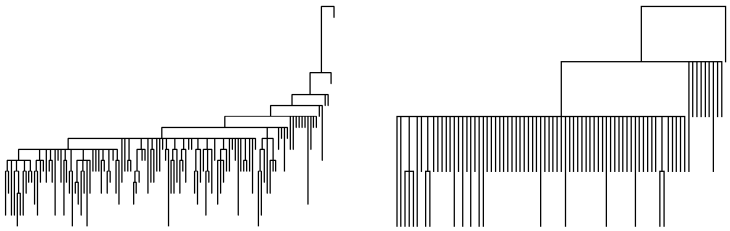
Fig. 6. FHTs based on EMD-1032: minimum component size of 20, pruned by subtree height parameter (a) $h = 4\Delta$, (b) $h = 20\Delta$, where $\Delta = (f_{\max} - f_{\min})/128$.

3.5 Elimination of Short Edges from FHTs

Pruning by subtree height only helps near the leaves. Other parts of the tree will not be simplified unless one chooses a very high value of h to prune by, in which case one could lose much valuable information.

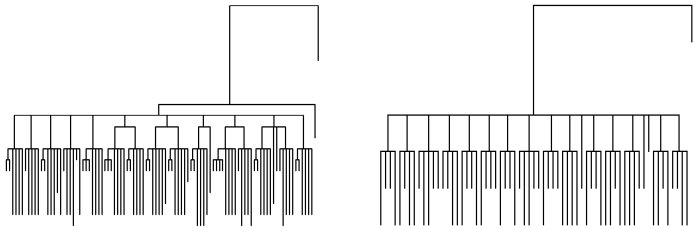
We have developed another method of simplifying FHTs that can be applied after pruning by subtree height, and which does not have this shortcoming. We call this method *elimination of short edges*, because it essentially eliminates edges *anywhere* in the tree that are not longer than a positive parameter ϵ. Figs. 7 and 8 show its effect on the FHTs of Figs. 5(a) and 6(a), for two different values of the parameter ϵ.

Simplification of an FHT T by elimination of short edges can be accomplished by calling SIMPLIFY(root(T), root(T), ϵ), where root(T) is the root of T and SIMPLIFY() is defined as follows:



(a) (b)

Fig. 7. FHTs based on EMD-1006: minimum component size of 25, pruned by subtree height parameter $h = 4\Delta$ and short edges eliminated by parameter (a) $\epsilon = 2\Delta$, (b) $\epsilon = 9\Delta$, where $\Delta = (f_{\max} - f_{\min})/128$.



(a) (b)

Fig. 8. FHTs based on EMD-1032: minimum component size of 20, pruned by subtree height parameter $h = 4\Delta$ and short edges eliminated by parameter (a) $\epsilon = 2\Delta$, (b) $\epsilon = 9\Delta$, where $\Delta = (f_{\max} - f_{\min})/128$.

```
SIMPLIFY(Vertex v, Vertex r, float ε):
    foreach child c of v do
        if L(c) > L(r) + ε then
            if r ≠ v then make r the parent of c;
            SIMPLIFY(c, c, ε);
        else
            SIMPLIFY(c, r, ε);
            Remove the vertex c;
```

We can give a non-recursive characterization of the effect of this simplification method. Let T be any FHT for an image f. We define an ϵ-*acceptable simplification* of T to be an FHT T' for f that satisfies the following conditions:

1. Vertices$(T') \subseteq$ Vertices(T), and the functions $L_{T'}$ and $D_{T'}$ are the restrictions to Vertices(T') of the functions L_T and D_T.
2. For all $p, c \in$ Vertices(T') such that p is the parent of c in T', $L(c) > L(p) + \epsilon$.

Let $\leq$ denote the partial order, on the set of all ϵ-acceptable simplifications of T, such that $T_1 \leq T_2$ if, and only if, each vertex in $\text{Vertices}(T_1) \setminus \text{Vertices}(T_2)$ has an ancestor in T that lies in $\text{Vertices}(T_2) \setminus \text{Vertices}(T_1)$. Then the FHT that is produced from the FHT T by $\text{SIMPLIFY}(\text{root}(T), \text{root}(T), \epsilon)$ is the ϵ-acceptable simplification of T that is maximal with respect to $\leq$.

A consequence of the fact that elimination of short edges affects all parts of the tree is that, for similar values of the parameters ϵ and h, elimination of short edges will typically remove many more vertices than pruning by subtree height.

4 Discussion

EM reconstruction need not preserve the geometry of the specimens under study. The foreground history tree (FHT) of a 3D gray-valued voxel-based image is a useful descriptor that is insensitive to topology-preserving transformations.

We have presented parametric methods of simplifying FHTs that remove artifacts due to noise, while preserving what seem to be essential spatial properties of the specimens. The simplified FHTs of different specimens obtained from a molecular database capture some of the structure in those specimens. For example, the FHTs in Fig. 8 of the helical structure shown in Fig. 2 contain a vertex from which many similar-looking subtrees are descended.

FHTs provide a potentially useful way of discretely representing essential aspects of the shape of a complicated object. Thus we believe that they can be used in methods of querying macromolecular databases.

Acknowledgments

This work was supported by the National Institutes of Health Grant HL070472. Our interactions with P. L. Combettes, J.-M. Carazo, R. Marabini, E. Garduño, S. H. W. Scheres, and I. Montealegre were helpful in preparing this manuscript.

References

1. Rosenfeld, A.: Three-dimensional digital topology. Information and Control **50** (1981) 119–127
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge, MA, USA (1990)
3. Carr, H., Snoeyink, J., Axen, U.: Computing contour trees in all dimensions. Computational Geometry: Theory and Applications **24** (2003) 75–94
4. Rosenfeld, A., Kak, A.C.: Digital Picture Processing. Academic Press, New York, NY, USA (1976)
5. Rawat, U., Zavialov, A.V., Sengupta, J., Valle, M., Grassucci, R.A., Linde, J., Vestergaard, B., Ehrenberg, M., Frank, J.: A cryo-electron microscopic study of ribosome-bound termination factor RF2. Nature **421** (2003) 87–90
6. Hoenger, A.: A new look at the microtubule binding patterns of dimeric kinesins. Journal of Molecular Biology **297** (2000) 1087–1103
7. Felsenstein, J.: PHYLIP - phylogeny inference package (version 3.2). Cladistics **5** (1989) 164–166