Review on the submission titled
"Design and Implementation of the Three-Dimensional Autonomous Leaves Data Structure"

The authors describe a three-dimensional data structure for storing and modifying a hierarchical space division. It consists of regular hexahedra that are obtained by isotropic refinement of one root cube. The refinement may be adaptive in the sense that none, some, or all of the eight children of a parent cube can be refined in turn. This principle can be applied recursively, effectively creating a non-overlapping space division where neighbor cells may have size differences by factors of any power of two. The proposed data structure explicitly stores the connections between neighbor cells across faces. Edge and corner neighbors are not stored.

The data structure is modeled using cells that are linked with pointers. The cells themselves are stored in a doubly linked list as a linear octree, meaning that only leaves are stored but no direct or indirect parent cells. The authors choose a Hilbert curve variant for their ordering. Every cell also stores six pointers for the face neighbors. When neighbors of different sizes meet, transition nodes are inserted that hold five pointers, one to the large face on one side and four to the smaller faces on the other. Transition nodes can be linked to arbitrary depth to accomodate higher power-of-two size differences, mapping 1:4, 1:16, etc. face correspondences between cubes on either side. The paper does not discuss algorithms for size balancing or parallelization.

All links are implemented using pointers in both directions; the transition nodes are doubly linked in this sense. Consequently, the paper consists predominantly of algorithms that change the mesh by refinement or coarsening and update the various kinds of bidirectional links. The authors close with numerical results for a finite volume method applied to a linear parabolic PDE modeled after cardiac electrophysiology.

The paper presents the author's work using detailed algorithms. This is helpful for the reader to understand the author's method, but carries the risk of excessively dwelling on rather obvious thoughts and steps. Many algorithms strike the reviewer as redundant and verbose, and not providing content that qualifies as relevant research. I compliment the authors on defining a consistent and fairly intuitive storage concept, and on developing a working set of methods for managing the data structure. However, besides the introduction of the transition nodes I can find little content that presents new ideas in terms of effective adaptive-octree algorithms. I will try to point out some major issues that I see with the current version of the paper.

1. Use of pointers.
The authors use pointers to connect their cells, which are C++ objects. The proposed cell structure requires at least 96 bytes plus the C++ overhead for class instantiation. While this concept is clean from a computer science perspective, the priority in computational mathematics is on performance. Lookup arrays and ordered indices would cut the storage requirements in half. They alse offer the conceptual advantage of exploiting the integer ordering for encoding the sequences of cells, faces, etc. relative to each other. (In fact, such correspondence is the cornerstone of several existing state-of-the art octree codes.) This comment extends to the transition nodes: Using integer lookups would provide an implicit ordering of the four connections on the

small
side, while the authors use a rather suboptimal 4-way switch statement in their
algorithms.

2. Cleanup algorithms
The authors always create transition nodes in refinement and derefinement, and provide separate algorithms for cleaning up redundant pairs afterwards. They reason explicitly that their method is simpler and less complicated, but I must
say that I disagree. I would imagine that enforcing the invariant of non-redundant transition nodes at every point in the algorithms would lead to simpler and faster code, not mentioning the elimination of two rather long-winded algorithm descriptions. Using the Hilbert index for the ordering of faces in the transition nodes would also eliminate some redundancy.

3. Hilbert curve
The authors use a modified Hilbert curve for ordering the leaves of the octree.
They store the branch number down the Hilbert tree at every level, thus avoiding the O(depth) transformation of the Hilbert index, which strikes me as a nice idea. On the other hand, the authors do not seem to be aware of the fact that the classical Hilbert curve has long been used in adaptivity. Why do they introduce their own concept instead of using the classical curve? How is it different? The authors do not provide any reasoning for their choice (which is certainly not more symmetric than the Hilbert curve) and why it should be used. Without a precise mathematical definition, the author's approach remains arbitrary and unidentifiable to the reader. Furthermore, the authors do not seem to make the connection between the bunch number they store,
the cell coordinates, and the Hilbert index, all of which are bijectively related to each other. The tables I-III might quite possibly be replaced by operating on the Hilbert index alone.

There are also a number of other issues, described as follows.

4. Some choices do not appear to be thought through. For example, the pattern used in Fig. 16a is not present in Fig. 15 and thus illegal by context.

5. The authors do not cite any references to research on pointer-based and linear octrees, nor to space filling curves and adaptive mesh refinement in general.

6. Many figures are lacking information. There are no coordinate systems provided with the graphics, thus they carry no precise information. Often, no definition is given of the orderings that are chosen.

7. In algorithm 11, there is an off-by-one error on line 8. It also processes the first cell twice due to an error in the loop logic. Line 14 would be more useful before line 13. The cell-after-bunch logic in lines 15--17 is not strictly needed and could be simplified. The decision in line 5 requires a sub-algorithm that could be fused with the loop in lines 8-10. Algorithm 18 seems especially inefficient since a 12-way switch statement is clearly slower than a non-redundant array lookup.

8. It is not clear what the meaning of Fig. 7 is. Pictures are very helpful for illustrating mathematical concepts, but cannot replace precise definitions and mathematical arguments.

9. The runtime of the tree algorithms is not reported. In discussing the

numerical application, no error or runtime analysis is provided.