Title
Multiresolution Cutting-Plane Visualization Based on an Octree

Authors
Pinskiy, Dimitriy V.
Ahern, S.
Brugger, Eric S.
et al.

Publication Date
2000

Peer reviewed

Multiresolution Cutting-Plane Visualization Based on an Octree

Dmitriy V. Pinskiy*, Sean Ahern†, Eric Brugger†, and Bernd Hamann*
(pinskiy@cs.ucdavis.edu, ahern@llnl.org, brugger1@llnl.org, hamann@cs.ucdavis.edu)

Cutting Plane through Sphere and Skull Data Sets. Red Polygon is ROI

1 Introduction

As data sets have grown in size to several million data points, considering finite element/difference simulations, the extraction of just a single cutting plane can become prohibitively expensive. We address this problem by presenting a technique that is based on a multiresolution approach.

Our goal is the efficient generation of cutting planes through extremely large data sets, allowing a user to slice volumetric data sets at arbitrary locations and in arbitrary orientations in real time. Within a specific cutting plane, a user can select a region of interest (ROI), where a higher rendering resolution is desired. The presented algorithm supports not only efficient cutting-plane rendering and local refinement but also permits moving an ROI in a cutting plane and allows ROI to be bounded by any convex polygon with arbitrary number of edges.

2 Related Works

The underlying data structure that we employ in is an error-controlled octree. Octrees permit the hierarchical representation of space, where a cube-like region is split into eight children. This data structure is described by Samet [4].

The input for our method is a set of points in space with associated function values ("scattered data"), for which we construct an octree. First, we compute the bounding box of the set of all given points. This bounding box is then subdivided in octree nodes until each leave node in the octree satisfies a specific termination criterion. We consider two types of termination criteria: (i) a node is no longer split when the number of original points lying in it is smaller than some threshold number N or (ii) a node is no longer split when the error between the function values of the original points inside the node and an average function value associated with the node is smaller than some error threshold E. (A similar error-controlled approach has been developed by Laur and Hanrahan, but their driving application is volume rendering [2].) By varying the values N and E, we enforce further subdivision of octree cells through which most likely the cutting plane will pass and of cells that contain an ROI. One of the main purposes of our octree construction is the fact that data points are spatially sorted using a data structure with extremely low storage requirements. The octree also allows us to find intersections with a cutting plane efficiently. (We use the intersection algorithm described by Muller and Heines[3].)

Our techniques is also related to the idea of constructing a "data pyramid," i.e., a hierarchy that increases the precision of an approximation by increasing the number of cells while shrinking cell sizes [1].

*Center of Image Processing and Integrated Computing, Department of Computer Science, University of California, Davis
†Lawrence Livermore National Laboratory

After the given scattered data points have been organized in an octree, a user can specify a cutting plane, and a set of tiles/polygons covering the cutting plane is rendered. Tiles are rendered according to function values. Typically, a user has to generate a large number of cuts before he/she has identified a specific cut of interest – a main region. Thus, it is imperative that the tiles making up a cutting plane be generated most efficiently. Our recursive algorithm works as follows: We consider the coordinates/extension of a node (a cube) and check whether the cutting plane intersects the node's bounding box. If it does not, there is no intersection between the plane and the node, and nothing is rendered. Otherwise, we check if the node whose bounding box we consider is a leaf. If this is the case, we clip the plane against the bounding box and associate the resulting tile with the function value of the node and insert the tile into the set of output tiles to be rendered. If it is not a leaf, then it is a branch, and we recursively apply the algorithm to every child.

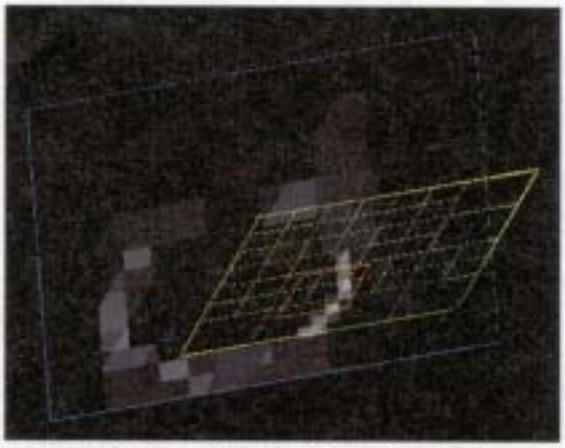
One of the important advantages of this algorithm is that we do not have to explore "useless" branches — branches that are not intersected by a cutting plane. By processing the scattered data using an octree approach, we achieve logarithmic complexity. Another advantage of this approach is that it simplifies performing local refinements – which is enabled by simply replacing certain tiles with smaller ones to achieve more rendering accuracy in a localized and adaptive fashion.

3 Region-of-Interest Construction

After a main region has been selected, a user can specify an ROI in the cutting plane for better-resolution rendering. To accomplish that, we perform these operations:

- Locate ROI — Determine which branches of the octree should be "expanded."
- Expand these branches.
- Repeat the clipping algorithm for the newly generated octree nodes to generate smaller and more precise tiles.
- Update the set of tiles for rendering.

We provide a user with only a low-resolution representation of a main region. Therefore, visually inspecting the main region, the user is able to make only a best guess about locating an ROI in a cutting plane. Once a specified ROI is rendered at a higher resolution, and a user might want to further adjust the location of the ROI by moving it within the cutting plane. It might require several iterations before a user finds the ROI he/she really is interested in. This "navigation" step can become time-consuming if one uses a naive approach. We have developed an approach that solves this problem efficiently: We employ an approach related to caching. We assume that a user's first guess for placing an ROI is good but not correct; we construct a quadtree that sorts the tiles/polygons near and inside the ROI in the current cutting plane. We have to build this quadtree only once. After that, we do not need to traverse the entire octree every time when a user moves an ROI inside some region around the initial ROI because all needed polygons are already sorted via the quadtree structure.



Rendering of Cutting Plane (Main Region) and ROI Surrounded by Quadtree

References

[1] L. De Floriani. A Pyramidal Data Structure for Triangle-Based Surface Description. *IEEE Computer Graphics and Applications*, 9(2):67-78, 1989

[2] D. Laur and P. Hanrahan. Hierarchical Splatting: Aprogressive Refinement Algorithm for Volume Rendering. *Proceedings of SIGRAPH 91*, pp. 285 - 288, July 1991

[3] T. Muller and E Heines. Real-Time Rendering. *A K Peters*, Natick, Massachusetts, 1999

[4] H. Samet. Applications of Spatial Data Structures. *Addison-Wisley*, Reading, Massachusetts, 1990