Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects

Douglass Davis, William Ribarsky, T.Y Jiang, Nickolas Faust, and Sean Ho
Graphics, Visualization, and Usability Center
Georgia Institute of Technology
{dougd, ribarsky, jiangf}@cc.gatech.edu, nick.faust@gtri.gatech.edu, sean@arl.mil

Abstract

This paper presents results for real-time visualization of out-of-core collections of 3D objects. This is a significant extension of previous methods and shows the generality of hierarchical paging procedures applied both to global terrain and any objects that reside on it. Applied to buildings, the procedure shows the effectiveness of using a screen-based paging *and* display criterion within a hierarchical framework. The results demonstrate that the method is scalable since it is able to handle multiple collections of buildings (e.g., cities) placed around the earth with full interactivity and without extensive memory load. Further the method shows efficient handling of culling and is applicable to larger, extended collections of buildings. Finally, the method shows that levels of detail can be incorporated to provide improved detail management.

I. Introduction

We have recently shown how intent and perception can be used to significantly improve out-of-core visualization of large scale terrain [1]. This paper demonstrates that these techniques can be generalized and applied to heterogeneous data. Although we deal here with geospatial objects, there is no reason why one could not employ a similar framework for time-dependent, 3D data. The importance of the present paper is that it lays out a general set of methods for effective out-of-core visualization and then applies them to discrete, static objects, such as buildings, that are quite different from the continuous terrain studied previously.

In this paper we show how our techniques can be applied to large collections of buildings, several hundred or more in each collection, that may be distributed at many spots throughout the world. Efficient out-of-core visualization methods permit timely paging of building data in support of real-time navigation. Our approach uses an efficient quadtree hierarchical structure appropriate for global terrain [2] at the top level followed by methods appropriate for 3D object detail management at lower levels. The quadtree level at which the change-over occurs from the quadtree hierarchy to the 3D object management structure will depend on the nature of the objects. The levels will be different, for example, for collections of buildings of a certain range of sizes versus vehicles. We provide a basis for determining this level.

II. Related Work

We have considered how out-of-core visualization applies to terrain datasets [1]. We found that a combination of out-of-core visualization, which tends to focus on 3D data, and visual simulation, which places an emphasis on visual perception and real-time display of multiresolution data, results in interactive terrain visualization with significantly improved data access and quality of presentation. Ref. 1 considers only terrain whereas the present paper generalizes to collections of 3D objects and provides criteria for detail management based on density and size distribution of objects.

Over the years there have been several methods that have been developed and used to cope with large amounts of data [3]. In early research, George and Rashwan used auxiliary storage methods to solve finite element problems [4]. Liu [5] applied an out-of-core multifrontal method for sparse factorization.

Recently there has been work that addresses interactive visualization of very large, out-of-core datasets. From what has been done so far, it is clear that application-control and domain-dependent data organization are essential to achieving good performance. Relying on system virtual memory, for example, frequently results in thrashing and abysmal performance. Ueng et. al. [6] apply an application-controlled segmentation approach to out-of-core visualization. They spatially and hierarchically partition the dataset into an octree and load only needed segments. One problem with their approach is how to determine segment boundaries. Cox and Ellsworth present application-controlled demand-paging [7], in which the system knows something about what data are needed and when. By considering operating system memory management, they minimize thrashing. Zyda and co-workers [8] came up with a hierarchical quadtree data structure by evenly subdividing data into square quadnodes and rendering with regular grid polygonalizations. Based on this regular grid they develop a paging method that takes into account the viewpoint and speed of the user. Quite recently Chiang et. al. [9] have developed an interactive technique for out-of-core isosurface

extraction from volume data. They developed a meta-cell technique for partitioning the original data and an indexing scheme for efficiently making isosurface queries into the metacells, which reside on disk, to bring in the appropriate data for constructing the isosurface. All the above out-of-core techniques consider the paging of continuous volumetric or terrain data. None considers application to collections of discrete 3D objects, as we do here.

The need to handle scalably large collections of buildings has come to the fore because there are improved methods to extract such data [10] and because applications requiring accurate display of urban data (such as emergency response, urban planning, or urban warfare) are growing in importance.

III. Placing 3D Objects in a Geospatial Hierarchy

The forest of quadtrees has proven capability for handling global terrain [1, 2]. When considering buildings or other geospatial collections of objects, we would like to build on this structure, at least at the top level. If we have several cities in our global dataset, we want to quickly determine that the viewer is navigating towards, say, Los Angeles and does not need detailed data for Bombay, Houston, or even San Francisco. Further, as the user navigates the greater LA area and gets close enough to view individual blocks of buildings in, say, Anaheim, we don't want to page in and individually test buildings from Pasadena.

On the other hand, when the user can discern individual buildings or groups of buildings, it is more appropriate to use methods focused on managing detail for the individual buildings or groups. The question is: at what level should this change-over be made? We present below an analysis of when this should occur. In the following we show how large collections of buildings or other objects can efficiently fit into the geospatial hierarchy. A full exposition of detail management for the groups of buildings is beyond the scope of this paper, but we show where such methods would fit and implement some simple techniques.

The top level hierarchy is shown in Fig. 1. The earth is divided into 32 zones, with each zone containing a quadtree [2]. For the buildings or other objects the quadtree goes to a certain level after which there is a non-quadtree detail management scheme. What should this level be? Let us assume a quadcell of side L_q (see Fig. 2.). Now let's assume an object with maximum dimension L_o. If $L_o < L_q$, then only the 8 surrounding quadcells might contain objects that would extend into a central quadcell. In fact if we divide the central cell into 4 quadrants and further assume that all objects are placed in the quadcells that contain their centers, the maximum number of quadcells whose objects could overlap this quadrant would be 4, the central cell plus the 3 nearest quadcells to the quadrant. Our approach is to go as deeply as possible using the very efficient quadtree but not to permit more than 4 linked cells for considering overlapping objects, since there will be increased overhead from keeping track of the links and from having to consider all the objects in the linked cells. The collection of objects in the linked cells would have to be considered in view frustum culling, collision detection, and many other operations. Thus we choose cells for which $L_o < L_q$ but these dimensions are close in size. For buildings in an urban setting, L_o could be about 50 M, the dimension of a typical city block. If a typical building is the size of a house, say 10 M on a side, we would have to consider no more than 100 or so buildings for 4 linked cells. Note that occasionally we will have to consider linked cells that could be in multiple quadtrees.

However, what about extended objects such as a stadium, the Pentagon, the Vehicle Assembly Building at the Kennedy Space Center, or very large objects created via detail management? The latter might require considering several blocks as one object. If we use the largest such object in the database to determine the leaf node level of the quadtree, we might also end up with cells containing several hundred smaller objects. To obviate this problem we have discrete representations of such large objects at successive levels of the quadtree, each representation carrying its own list of linked cells. Thus if we flew from outer space down towards an urban area, we might first see a phototextured shape representing the downtown area, which would then be replaced by more detailed shapes representing collections of blocks and tall landmark buildings, and finally these would be replaced by shapes for individual buildings. Although the present system switches between discrete representations, one can imagine a more sophisticated process with more continuous switching of detail.

Since such large objects are relatively small in number, we can handle them reasonably efficiently even though they carry their own lists and descriptions. Also such representations usually occur before the appearance of large numbers of smaller objects. When considering small buildings at urban densities, however, we use our more automated and compact linked cell mechanism.

IV. A Hierarchical Structure for Optimized Paging

As discussed above our global structure is divided into 32 zones, each $45^o \times 45^o$ [1,2]. Each quadrant has its own quadtree; all are linked so that objects or terrain crossing quadrant boundaries can be rendered correctly. To improve performance, the system is divided into multiple threads that can run in parallel. In particular, there is an independent rendering thread,

which has a "triple buffer" of display lists. One of the display lists contains what the renderer is currently drawing, one is used by the scene manager to buffer graphics commands, and the last contains data that are ready to be displayed.

Both the object and terrain paging threads have a server and manager. The object server loads pages from disk while the manager decides which cells should be loaded (taking into account user viewpoint and navigational speed) and passes it along to the scene manager. The object server and manager communicate with the scene manager and the rest of the system via shared caches, so that communication is limited to small request messages and acknowledgments. This communication path supports a demand-paging approach such as that of Cox and Ellsworth [7]. When data are needed for a node in the quadtree, the scene manager allocates space in the shared cache and sends a message via a shared memory priority queue to the object manager. Message priorities in this queue are changed dynamically according to the importance of the associated request as determined by the terrain manager. Thus, requests that gradually become less important sift towards the end of the queue and get serviced only when no higher priority requests remain in the queue.

The underlying disk management system has a file structure with files aligned with the quadnodes in the set of linked quadtrees. Put together, all this makes the object and terrain visualization system quite scalable. Tens to hundreds of gigabytes of data may be made available for visualization, either locally or remotely.

Object Page Scheduling. We have found that the above page priority procedure sometimes falls short when handling global data. Users of such data frequently fly quickly from a global view where the terrain elevation and imagery data are at 8 Km resolution to views close to the ground where the data are at 1 M resolution or higher, and there may be hundreds or more buildings in view. If the user flies in too fast, the traversal of linked quadtrees by the terrain manager falls well behind the user's navigation. The process can stall in this case, and the pages for the scene currently in view can take quite long to arrive.

Unfortunately the system cannot just jump to the appropriate position in the quadtree. The quadtree has to be traversed to get important properties information, especially quadcell linking data but also geospatial bounding boxes and other data, that are necessary to determine if the object data should be displayed or not. To address this problem we created a modified version of the separate set of indexing trees used in the terrain paging system [1]. This separate structure provides properties information but is lightweight so it can be traversed quickly. Large segments of the indexing trees reside in main memory for fast access. With the flexibility of this scheme we can skip one or more levels before paging in object data. A predictive mechanism is instituted based on user navigational speed and viewing direction to help predict where the terrain manager should skip.

Since the scene manager is receiving continuous updates from the user via the user interface, it can use these in its requests to the object manager. The scene manager can, for example, expend more detail on buildings or other objects in the center of the screen.

IV. Building Display Results

We present results for a collection of 471 textured buildings on our global terrain database. The terrain has worldwide coverage at 8 Km, U.S. data at 1 Km, Georgia at 100 M, and several insets ranging from 10 M to 0.5 M. The total size of the terrain alone is 1.5 GB. The 471 buildings are for downtown Atlanta and are thus placed on top of high resolution terrain data at 1 M resolution. When buildings are added without the hierarchical paging structure, the frame rate drops from 15-20 frames per second to 7 frames per second, even on the SGI Infinite Reality. We have lately developed faster object rendering (not used in the results of this paper), but frame rate still slows considerably when buildings are loaded.

The frame rates indicated in Figs. 3-7 (the first number in the lower left corner) were obtained on an SGI Infinite Reality with 4 R10000 processors, 1 GB of memory, and 27 GB of disk. All results depicted were obtained on continuous fly-in or fly-out with no pauses (except for the skyline in Fig. 7 where there was a pause during switch from fly-in mode to parallel to the earth mode). The fly-in rate was such that one could navigate from a whole earth view at over 10,000 Km to 10 M above the earth (in an area with high resolution data) in 30 seconds.

The hierarchy permits a simple but effective detail management process. Object pointers are loaded at the quadlevel determined as described in Sec. III. Accompanying the pointer is the object location and a bounding dimension obtained from the largest dimension of the object bounding box. Using the viewpoint , object location, and bounding dimension, the system finds a maximum size for the object in screen space in terms of pixels. (See Fig. 2.) For the results presented in Fig. 4, any object with bounding dimension greater than 1 pixel is loaded and displayed. This conservative approach works well; the largest buildings in the database don’t appear until the user navigates to about 160 Km away from them. No buildings are paged in until this point, and frame speed does not slow perceptibly until the user is closer and more buildings are paged in. (See Figs. 4, 6 and 7.) This shows the efficient scalability associated with our hierarchical paging process. Buildings in groups of any size could be placed all over the earth and neither performance *nor* memory load would be affected till the user is relatively close to them. Of course we could also apply a more

sophisticated boundary dimension that would take into account orientation of the object.

Note that the hierarchical paging works for any objects including those representing collections of buildings. We have used this to do simple detail management. We have preprocessed the buildings to produce 3 levels of detail, a shape representing the extent and simplified skyline for the whole collection of buildings, and two other sets of objects representing portions of the building collection. These objects were placed at appropriate (higher) levels of the quadtree, as shown in Fig.2. A threshold of 1 pixel was set for these objects and 10 pixels for the individual buildings. When the underlying buildings are paged in the higher level objects disappear. As shown in Fig. 6, this permits significantly higher frame rates until the viewer is much closer to the buildings than before, while still retaining detail about the location and shape of the building collection.

Additionally the hierarchy aids in culling when the viewer is panning or flying around the buildings. Comparing Figs. 6 and 7 with Fig. 5 shows this clearly as the former have lower frame rates because some of the buildings are culled. Buildings not close to the view frustum are not even paged in. This makes feasible the handling of extended building collections that are much larger than the one considered here. In fact, we are planning to test such an extended collection by replicating the current set of buildings several times side-by-side.

V. Conclusions and Future Work

We have presented results for real-time visualization of out-of-core collections of 3D objects. This is a significant extension of previous methods and shows the generality of hierarchical paging procedures applied both to global terrain and any objects that reside on it. Applied to buildings, the procedure shows the effectiveness of using a screen-based paging *and* display criterion within a hierarchical framework. The screen-based tolerance is adjustable and directly provides a control for image quality.

The results demonstrate that our method is scalable since it is able to handle multiple collections of buildings (e.g., cities) placed around the earth with full interactivity and without extensive memory load. Further the method shows efficient handling of culling and is applicable to larger, extended collections of buildings. Finally, the method shows that levels of detail can be incorporated to provide improved detail management.

We plan to extend object detail management by providing more accurate levels of detail, as depicted in Fig. 2 above. Our goal will be automatic procedures that can be applied during preprocessing. We will also consider dynamic control of levels of detail for building and other features (such as streets and trees) beyond the hierarchy, as shown in Fig. 1.

Acknowledgments

This work was performed in part under contract DAKF11-91-D-004-0034 from the U.S. Army Research Laboratory. We would like to thank Larry Tokarcik and his team at the Army Research Laboratory for supplying data used in this work.

References

1. Douglass Davis, T.Y Jiang, William Ribarsky, and Nickolas Faust. Intent, Perception, and Out-of-Core Visualization Applied to Terrain. Report GIT-GVU-98-12, pp. 455-458, *IEEE Visualization '98*.
2. Peter Lindstrom, David Koller, William Ribarsky, Larry Hodges, and Nick Faust (1997). An Integrated Global GIS and Visual Simulation System. Report GIT-GVU-97-07.
3. N.M. Brenner. Fast Fourier Transform of Externally Stored Data. *IEEE Trans Audio and Electroacoustics* 17(2), pp. 128-132 (1969).
4. A. George and H. Rashwan. Auxiliary Storage Methods for Solving Finite Element Systems. *SIAM J Scientific and Statistical Computing* 6(4), pp. 882-910 (1985).
5. J.W.H. Liu. On the Storage Requirement in the Out-of-Core Multifrontal Method for Sparse Factorization. *ACM Trans. Math. Software* 12(3), pp. 249-264 (1986).
6. S.K. Ueng, C. Sikorski, and K.L. Ma. Out-of-Core Streamline Visualization on Large Unstructured Meshes. *Transactions on Visualization and Computer Graphics* 3(4), pp. 370-379 (1997).
7. M. Cox and D. Ellsworth. Application-Controlled Demand Paging for Out-of-Core Visualization. Proceedings, *IEEE Visualization '97*, pp. 235-244 (1997).
8. J.S Falby, M.J. Zyda, D.R. Pratt, and R.L. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics* 17(1), pp. 65-69 (1993).
9. Y. Chiang, C. Silva, and W.J. Schroeder. Interactive Out-of-Core Isosurface Extraction. *IEEE Visualization '98*, pp. 167-174 (1998).
10. Tony Wasilewski, Nickolas Faust, and William Ribarsky. Semi-Automated and Interactive Construction of 3D Urban Terrains. Accepted for publication, *Proceedings of the SPIE Aerospace/Defense Sensing, Simulation & Controls Symposium* (1999).