Intent, Perception, and Out-of-Core Visualization Applied to Terrain

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

Abstract

This paper considers how out-of-core visualization applies to terrain datasets, which are among the largest now presented for interactive visualization and can range to sizes of 20 GB and more. It is 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. Further, the visual simulation approach provides qualities that are useful for general data, not just terrain.

I. Introduction

Interactive visualization of big data presents a significant challenge, but also offers the possibility of major rewards. Visualization is often the first step towards analysis and understanding. We must first "look at" the data before we can even know what to do next. Conversely, if we cannot even visualize the data, we cannot effectively explore and discover. We can only perform peripheral analysis; much of the data remains untouched.

To meet this challenge, out-of-core visualization approaches have lately been developed [1-3] that allow one to display data even when the dataset is much larger than memory. These approaches attempt to organize the data in such a way that they can be transmitted from disk, either local or networked, to memory in time to support interactive visualization. To the extent that these methods are successful, they increase visualizable data by 1, 2, or more orders of magnitude and make possible the interactive exploratory analysis of many simulational and observational datasets.

In this paper we consider how out-of-core visualization might be applied to terrain datasets, which are among the largest now presented for interactive visualization. For example, we provide interactive flythroughs of datasets ranging to 20 GB in size; even larger datasets are possible. We find that this combination of out-of-core visualization, which tends to focus on 3D data [1-3], and visual simulation, which places an emphasis on visual perception and real-time display of multiresolution data [4,5], results in interactive terrain visualization with significantly improved data access and quality of presentation. This set of very large dataset visualization methods will also be of prime importance for visual simulation systems that handle not only terrains but large scale collections of moving and static 3D objects and 4D fields of data.

II. Related Work

The out-of-core problem is not new. Methods have long been developed and used to cope with large amount of data [6]. In early research, George and Rashwan used auxiliary storage methods to solve finite element problems [7]. Liu [8] applied an out-of-core multifrontal method for sparse factorization. In recent years, researchers have looked at disk storage I/O optimization. In this approach, disk storage is treated as another level in the memory hierarchy, below cache, local memory, and remote memories. Parallel programming models and software programs are used essentially to manage the movement of data between any two adjacent levels in a hierarchical memory system.

Until recently, however, there has been little work that directly addresses the problems of out-of-core visualization. But 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. [2] 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 [1], 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 [4] 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 developed a paging method that takes into account the viewpoint and speed of the user. Our approach applies to much deeper and multiple linked quadtree structures so that one can consider global terrains, rather than patches as in the Zyda approach. Further we apply a continuous level of detail method, instead of fixed levels, so that visual artifacts during paging are minimized. In all these cases, dataset paging and rendering are considered separately; there is little in the literature on adapting application-controlled segmentation for optimal rendering. This is one of the issues we address in this paper.

III. Applying Intent and Perception to Optimize Paging

For accuracy and speed, our terrain visualization system subdivides the globe into 32 quadrants, each $45^o \times 45^o$ [5]. Each quadrant has its own quadtree; all are linked so that 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 for each window (there can be multiple windows). 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. This scheme allows both the renderer and scene manager to run simultaneously without having to be synchronized.

The terrain paging thread has a terrain server and terrain manager. The terrain server loads pages from disk while the manager decides which terrain level of detail should be loaded (taking into account user viewpoint and navigational speed) and passes it along to the scene manager. The terrain 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 [1]. 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 terrain 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. This is important, as the paging rate during short bursts of requests is typically much lower than the request rate.

This paging model is quite flexible, as priorities can be assigned or changed quickly, based on a variety of criteria. These include, for example, giving lower priority to highest resolution pages when the user is navigating over the terrain above a certain speed or giving priority to pages in the center of the view frustum. 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 terrain visualization system quite scalable. Tens to hundreds of gigabytes of data are available for visualization, either locally or remotely.

Optimizing Paging

We have found that the above page priority procedure sometimes falls short when handling global data. Users of such data frequently fly 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. This is a change in page area scales of 8000 x 8000 or more! If the user flies in too fast, the traversal of linked quadtrees by the terrain manager falls well behind the user's navigation. In this case the paging can stall, taking 20 seconds or more to bring in high resolution pages. This happens even though the pages themselves are of modest size (each page is 70-75 KB), and the time to page in an entire scene is usually nearly a factor of 10 faster.

We did a detailed investigation and found that this problem occurred when the queues became so backed up that pages were being read in after they were passed in the navigation. Our straightforward scheduling scheme broke down in this instance. This was exacerbated by the problem that pages had to be read in to get important properties information they contained, such as geospatial bounding boxes and elevation delta values, that were necessary to determine if the pages should be culled and at what resolution they should be displayed [9]. To address this problem we set up a separate set of indexing trees that were connected to the properties information but were lightweight so they could be traversed quickly. (See Fig. 1). Large segments of the indexing trees reside in main memory for fast access. We use these trees and the properties information they contain to determine when pages should be paged in from disk. With the flexibility of this scheme we can skip one or more levels before paging in terrain elevation and image data. We have instituted a predictive mechanism based on user navigational speed and viewing direction to help predict where the terrain manager should skip. However, the skips cannot involve too many quadtree levels, because there will be too few vertices to complete high resolution triangles (since some vertices come from the skipped levels). This would cause gapping in the terrain fabric that could appear and disappear during navigation. We have determined that the maximum jump is 5 or 6 levels, and thus impose that constraint on the terrain manager. The result of these steps is significant improvement in time-to-display for pages, as discussed in the next section.

Since the scene manager is receiving continuous updates from the user via the user interface, it can pass these along to the terrain manager. This allows the paging process to respond to user intent and action, as we have seen, and also to make use of user perception. Our first perceptually-based implementation is to bring in pages first at the center of the view frustum, since this is where the user interest is likely to be. Previously, pages came in randomly in the viewing area, but with the new implementation, contiguous pages at the center of interest come in first. Thus the user need not wait for the whole scene to page in at high resolution, so this new paging scheme decreases *effective* time-to-display. Results are discussed in the next section. We have extended this concept to head-tracked implementations of the terrain visualization system on the virtual workbench. Here the direction of the user's head is also taken into account in determining which pages to display first. It would be straightforward to extend to eye tracking systems, providing an even tighter coupling to user actions.

We are considering other couplings between user actions and perception. For example, when a user is flying fast and low over terrain, the pages may be displayed at lower resolution than when she pauses. We are considering an implementation that constrains the manager to be at or above a certain level of the tree, based on user speed and height.

IV. Terrain Visualization Results

We present results for a global terrain database. It has worldwide terrain 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 dataset is 1 GB. The results presented here do not differ significantly using an appropriately configured system with local disks and up to ten or more GB of data. If we did not use paging at all and could insure that data would be brought in without thrashing, we would have to bring in a whole new scene whenever the memory limit of the machine was reached. This time would be of the order of the start-up time for the visualization system, about 3-5 minutes on an SGI Infinite Reality. For high resolution areas, this might occur after the user navigates only a few tens of miles. From Table 1 below, we see that our initial out-of-core visualization algorithm cuts this time by a factor of up to 100. In these timings, the machine used is either an SGI IR or a dual processor PC running Windows NT. The Infinite Reality has 4 R10000 processors, 1 GB of memory, and 9 GB of disk. The PC has two Pentium Pro 200 processors, 160 MB of memory, 4 GB of disk, and an Evans & Sutherland RealImage 3D graphics card. Thus we have tested machines at widely different levels of performance and memory capacity.

Our optimized paging, using the indexing quadtree that permits level skipping with prediction based on user speed and direction, produces a smoothing of the time-to-display as one navigates high resolution terrain, as seen in the rms deviations in Table 1. It also produces a 3-4 times improvement on the IR in average time-do-display. The optimized PC timings for Atlanta are higher than for NTC (National Training Center) because in the latter case more elevation nodes are skipped than in the former. Note that the PC shows 4-7 times improvement over the non-optimized case, presumably because of less contention over the data bus when fewer pages are sent. It also shows a much lower rms deviation for the same reason.

We also present in Table 1 results for networked paging on the IR. These are for a remote disk accessed via a connection that travels over a high speed link between two ethernets. Thus the data pass through routers. Even so, the paging is comparable to local paging, though the fluctuation is larger. This result indicates that interactive visualization can be retained even when using networked data.

Figs. 2 and 3 compare the non-optimized and optimized algorithms, at the same point after the same navigation process, for high resolution (1 M) mountain terrain from the NTC. This is at about the same point in the navigation process for which the timings were taken in Table 1. In the non-optimized case the user would have to pause here since the scene at this distance is too fuzzy to make out details. The pause might be up to 10 seconds on the IR and much longer on the PC.

In Figs. 4 and 5 we present views, after the same navigation process, with and without priority paging of the view center. The view is for 0.5 M urban Atlanta data a few hundred meters above the ground. We see that the view-centered image displays a coherent picture so that the user can interact and even continue to navigate. If we define the effective time-to-display as the time to get such a picture, it takes about 50% less time compared to non-priority paging of the whole image. Thus there is an effective doubling in the rate at which the user can navigate and interact.

Table 1

	Non-optimized	With skipping and prediction
To Atlanta (IR, remote)	11.75 ±2.58 sec	4.25±1.43
To NTC (IR, local)	13.81±0.95	3.36±0.16
To Atlanta (PC, local)	72.12±3.63	17.28±1.19
To NTC (PC, local)	70.35±14.53	10.08±0.51

V. Conclusions and Future Work

We have presented results for new out-of-core visualization techniques applied to very large scale terrain data. By combining techniques such as hierarchical data structures, appropriate page sizes, and demand-paging with visual simulation approaches, we have come up with improved techniques. These use prediction based on user actions and perceptually-based display based on user intent. Our results show that out-of-core visualization in a visual simulation framework produces results that permit interactive visualization and navigation even as datasets exceed tens of gigabytes. In addition, our approach provides interactive visualization even for remotely accessed data. Finally, we find that the methods are especially effective on systems with lower data I/O rates, such as PCs.

We believe the general techniques presented here are more widely applicable. It is already known how demand-paging and hierarchical structure can be used for out-of-core visualization of other types of data [1,2]. The use of a flexible, lightweight hierarchical framework for quick, prediction-based traversal can also be of significant help for other data. Further, a tight coupling of the user interface to the paging process will permit priority page selection based on user actions and intent such as described here. In fact, we are in the process of generalizing our terrain visualization system to handle large numbers of moving objects, buildings, and 3D data fields such as weather. All these data will be paged, and we expect to use the out-of-core visualization framework described here.

In addition, we expect to do further work on perceptually-based paging. This includes loading lower resolution pages when doing high speed flying close to the ground. We will also explore extending our database structure to include page priority information. With this information we can load pages based not only on perceptual factors but also on relative importance. We expect this to be especially useful for the paging of buildings and moving objects.

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 databases used in this work.

References

1. M. Cox and D. Ellsworth. Application-Controlled Demand Paging for Out-of-Core Visualization. Proceedings, *IEEE Visualization '97*, pp. 235-244 (1997).
2. 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).
3. D. Lane. UFAT: A Particle Tracer for Time-Dependent Flow Fields. Proceedings, *IEEE Visualization '94,* pp. 257-264 (1994).
4. 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).
5. 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, submitted to *Transactions on Visualization and Computer Graphics.*
6. N.M. Brenner. Fast Fourier Transform of Externally Stored Data. *IEEE TRans Audio and Electroacoustics* 17(2) pp. 128-132 (1969).
7. 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).
8. 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).
9. Peter Lindstrom, David Koller, William Ribarsky, Larry Hodges, Nick Faust, and Gregory Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. Report GIT-GVU-96-02, *Computer Graphics (SIGGRAPH 96)*, pp. 109-118 (1996).