A Distributed Memory Hierarchy and Data Management for Interactive Scene Navigation and Modification on Tiled Display Walls

Duy-Quoc Lai, Behzad Sajadi, Shan Jiang, Gopi Meenakshisundaram (M. Gopi), and Aditi Majumder

Abstract—Simultaneous modification and navigation of massive 3D models are difficult because repeated data edits affect the data layout and coherency on a secondary storage, which in turn affect the interactive out-of-core rendering performance. In this paper, we propose a novel approach for distributed data management for simultaneous interactive navigation and modification of massive 3D data using the readily available infrastructure of a tiled display. Tiled multi-displays, projection or LCD panel based, driven by a PC cluster, can be viewed as a cluster of storage-compute-display (SCD) nodes. Given a cluster of SCD node infrastructure, we first propose a distributed memory hierarchy for interactive rendering applications. Second, in order to further reduce the latency in such applications, we propose a new data partitioning approach for distributed storage among the SCD nodes that reduces the variance in the data load across the SCD nodes. Our data distribution method takes in a data set of any size, and reorganizes it into smaller partitions, and stores it across the multiple SCD nodes. These nodes store, manage, and coordinate data with other SCD nodes to simultaneously achieve interactive navigation and modification. Specifically, the data is not duplicated across these distributed secondary storage devices. In addition, coherency in data access, due to screen-space adjacency of adjacent displays in the tile, as well as object space adjacency of the data sets, is well leveraged in the design of the data management technique. Empirical evaluation on two large data sets, with different data density distribution, demonstrates that the proposed data management approach achieves superior performance over alternative state-of-the-art methods.

Index Terms—Management of computing and information systems, project and people management, life cycle, the computing profession miscellaneous ethics

1 Introduction

Interactive modeling capabilities are critical in designing virtual environments for several applications like urban planning and modeling, complex machinery modeling, or modeling of complex scenes and animated characters. Interactive navigation and modification of massive virtual reality (VR) models and environments are challenging due to two reasons. First, the size of the data required for rendering may be too large to fit in the memory. Second, interactive modifications cannot be done to gigantic 3D data sets stored on the hard disk, primarily because they mangle the data layout on the hard disk, which would affect the performance of later interactive navigation [24].

Hence, a non-redundant distributed data paradigm holds promise to mitigate the problems in interactive modification of large data sets, and the subsequent out-of-core interactive navigation, without focusing on load balancing in rendering. A cluster of storage-compute-display (SCD) nodes—where each node is self-sufficient, having its own storage, computation, and display capability—lends itself for an efficient integrated approach to both non-redundant distributed data management and interactive rendering.

Interestingly, an infrastructure of such a cluster of SCD nodes already exists in the display domain. Tiled multi-displays, either projection-based or LCD-panel based, already use an infrastructure consisting of multiple PCs, in which each PC drives a part of the display. Since today's commodity PCs come with powerful CPUs and GPUs, this same cluster of PCs can be considered as a cluster of compute-display nodes for computation and communication. Further, each PC is also configured with a very large secondary storage and main memory, which can turn each PC into the storage-compute-display node required for distributed interactive 3D modeling and rendering of the data set.

1.1 Main Contributions

In this paper, we design a new distributed data management system for interactive navigation and modification of massive 3D models on tiled displays. Our architecture enables multiple machines (including storage and computational machines) to behave like a single machine for the building and modifying a data set (which typically is a virtual reality scene), and multiple displays to behave like a single display for interactive navigation through the model. Our proposed distributed *data management* techniques are complementary to, and can be used along with, any state-of-the-art parallel and distributed *rendering* techniques proposed in the literature.

• *The authors are with the Department of Computer Science, University of California, Irvine.*
E-mail: {laid, bsajadi, sjiang4, gopi, majumder}@ics.uci.edu.

Manuscript received 20 Sept. 2013; revised 17 Sept. 2014; accepted 5 Nov. 2014. Date of publication 29 Jan. 2015; date of current version 1 May 2015.
Recommended for acceptance by A. Ynnerman.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2015.2398439

Specifically, we propose a distributed memory hierarchy for rendering applications on tiled displays. Further, we propose a data distribution technique for distributed secondary storage among all these participating nodes, in order to balance their data-servicing load for a walkthrough rendering.

Centralized control in distributed architecture is usually the bottleneck for the scalability of the system and also an impediment to interactivity.

Our *single level, loosely coupled distributed architecture*, with no central master, allows multiple storage-compute-display nodes to coordinate among themselves and evolve into a virtually unified single compute and display node. Thus, navigation and modification of the VR model is considered as a unified process, rather than a separate and disjoint one. While such a perspective is common when using single desktop PCs with small data sets, it has not been previously adopted when using clusters of SCD nodes, primarily because of the complexity of simultaneously handling non-redundant data partitioning and rendering across multiple machines and displays. Our single level architecture makes our system extremely scalable and flexible in massive model navigation and modification of virtual environments.

2 Related Work

Parallel rendering architecture. Although our work is related to data management and can be used along with any distributed rendering system, we will only discuss some of the literature in this paper. There exists a plethora of work in distributed rendering paradigms like SAGE, Equalizer, Chromium and Gigawalk [1], [7], [14], [20], where the rendering is distributed among machines, each of which is connected to a display node—essentially on a cluster of compute-display nodes. Prior sort-first or sort-middle systems [1], [14], [19] follow a pseudo-centralized data management architecture in which the entire data is duplicated at each node to reduce network load and avoid the movement of data. The data can be managed in the traditional triangle-soup format [1], [14] or can be pre-organized using any particular data structure, such as a graph [19], a space filling curve [4], [30], or an octree [3]. When there is a request from the application, this data structure can then be traversed to locate the requested data [13], [22]. The data management handled by such data structures, is very similar to a centralized system, where one central node would have the data, and would send it to multiple rendering nodes to render their part of the scene. The resulting frame would be stored on the rendering node until the central node tells the rendering node to display the frame.

Some prior work [7], [20] follows a sort-last architecture, although [7] can also operate in sort-first mode. In any sort-last architecture, setup of the display region and the rasterized frame dominates the rendering time. Further, for the same size of the data, the scalability of the system is dictated by the size of the frame buffer since the communication between the nodes depends on the frame buffer size. Hence, performance decreases if the size of the display is increased. In addition, an editing application needs back-indexing of the screen coordinate to geometric primitive, which is also prohibitive in sort-last architecture. Finally, the shipment of pixels to different display nodes for final sorting requires that if there is a minor change in the viewpoint, and hence a change in the image, the network will be used to sort and recompose the entire frame in all of the display nodes, even if the data did not change. To avoid all these limitations, [7], [20] are only used for navigation and not for editing. Further, the data is often duplicated across different rendering nodes, while a central node oversees the entire system and delegates the appropriate work to each render node. Note that [7] leaves the data distribution approach open to the application.

Recent work like the ones proposed by Erol et al. [8], and Moloney et al. [17] present different methodologies for load balancing of the rendering process which, often requires that the data be rendered on one node and displayed on another node. Our system differs from this because it primarily focuses on the efficient distribution of data, and each rendering node is coupled to its own display.

It is evident that a sort-first architecture [16] is most conducive for editing purposes, since it allows easy back-indexing. Therefore, along with our distributed data management system, we use a simple sort-first parallel rendering system in our experiment. The aim is to distribute 3D primitives, early in the rendering pipeline, to processors that can do the remaining rendering calculation. This leverages the low communication cost advantages of the sort-first architecture, because 3D primitives are only sent once to the rendering node, and is used by that node in rendering successive frames. Although some of the disadvantages of a sort-first architecture, related to complex data handling strategies, are adequately remedied by our novel data layout and management technique; others, including render-load imbalance, are not eliminated. Our work uses a form of distributed shared memory management that is similar to the ones proposed by Nieplocha et al. [18], Chapman and Heiser [5], and Fitzpatrick et al. [10]. Unlike the aforementioned work, the shared memory management is integrated into a memory hierarchy to improve the rendering performance of the system.

In summary, we present a new distributed data management method, which is loosely integrated with the rendering system. A simple parallel rendering method is also presented that takes advantage of the tiled display architecture and the underlying data distribution. It should be noted that our data management system is complementary to the rendering system; hence, any system that does not assume a specific data layout can use our data management system.

Data layouts. There is little prior work on interactive editing of large models and scenes, primarily due to the conflicting need for a good data layout for interactive rendering and imperative shuffling of data during the editing process. In order to achieve interactive rendering of large 3D walkthrough scenes, various acceleration techniques, like model simplification and cache coherent data layouts on hard disk, have to be used. When the scene is modified, however, it is difficult to rearrange the layout to maintain a consistent out of core data format or cache coherent data layout. Hence, the data organization on the disk degenerates quickly after modifications and edits of the data, which significantly affects the performance for interactive navigation. Sajadi et al. [24] have proposed solutions for interactive edits while maintaining interactive rendering speeds for Solid State

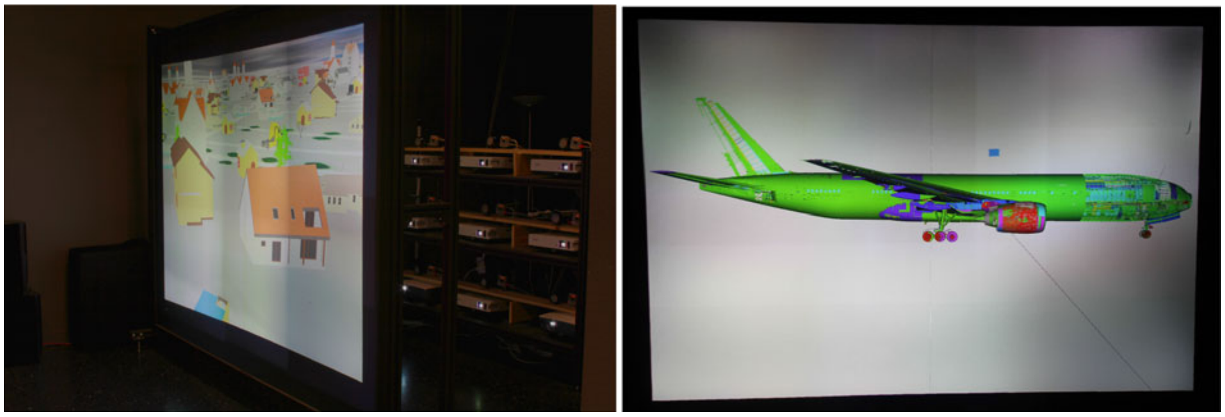


Fig. 1. Left: This shows a display of the City model on our 3×3 grid of storage-compute-display nodes, projected from a 3×3 multi-projector display. Right: A 20 GB Boeing 777 model, displayed on the 3 x 3 SCD grid.

Drives, however, only for single display system. Finally, extensive work have been done in data partitioning for interactive rendering on a single computer such as Far Voxels [11], Quick-VDR [31], and other cache-oblivious data layouts [3], [9], [15], [29]. Our work is complementary to all of the above work, since we do not propose a data layout method on a single computer, but propose a data distribution method on multiple computers. On individual machines in our cluster, any of the aforementioned data layout methods can be used to store the data allocated to that particular node by our data distribution algorithm.

Generating an optimized memory layout of the data is a computationally intensive process; therefore, it is only worthwhile to do for relatively stable data sets. If the data are truly dynamic and change every frame, then the cost of computing a layout outweighs the benefits since the layout is suitable only for one or a small number of frames. For such data sets, the layouts are computed only after the positions of the models are stabilized. We assume that the virtual environment data set undergoes changes that are found in a typical editing scenario. These models reside in the main memory during the editing operation, and the layouts are computed after the changes are committed.

Unlike existing distributed rendering schemes that duplicate the data, our scheme partitions the data. Such data partitioning has many advantages, out of which two are very relevant. (a) This leads to *data scalability* where the data size in each SCD node is small enough for interactive rendering and modification, even for very large 3D virtual environment models. (b) Since the data size in each node is small, it makes the data layout schemes less relevant for interactive navigation. Therefore, *efficient model modifications* can be done without the concern of expensive data layout recomputation or its impact on interactive navigation. Hence, our distributed data management approach, tied closely with the tiled display architecture, seamlessly addresses the various requirements of massive VR environments. It allows not only interactive visualization, but also interactive model editing during navigation, in an effective manner.

3 Distributed Memory Architecture

Fig. 2 shows the proposed distributed infrastructure and memory architecture for navigation and modeling, using a tiled cluster of SCD nodes. Each display node is powered by a compute node. Each compute node is equipped with a CPU, GPU, a secondary storage, and main memory. All compute nodes are connected via a network; in most cases, this will be a local area network (LAN). All physical connections between the computer, display, and network are depicted by the solid black lines in the figure. We assume that the displays of the SCD nodes are arranged in a grid. We assume that an adjacency graph defines the topology of the display grid and is stored at each SCD node. We consider an SCD node to be adjacent to another if their displays are spatially adjacent or partially overlapping. The adjacency graph can be automatically generated in this case at each SCD during camera based calibration and registration process using several prior techniques such as those proposed by Bhasker et al. [2], and Sajadi et al. [25], [26], [27]. Fig. 3 shows such an adjacency graph for a 3×3 array of nine projectors, the prototype system we use in this paper.

3.1 Memory Hierarchy

We denote the secondary storage of each SCD node as external memory (EM) and the local main memory as LC (local

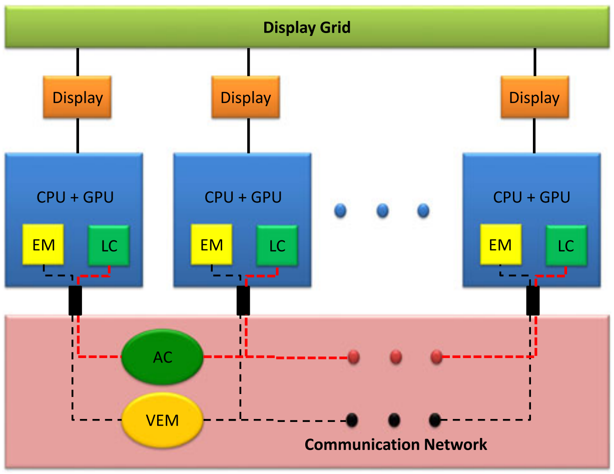


Fig. 2. Proposed distributed memory hierarchy. The distributed architecture utilizes a cluster of PCs to define a new memory hierarchy—from the fastest to the slowest—the Local Cache, adjacency cache, local external memory, and virtual external memory. When the data are not found in a faster memory, it is searched in and fetched from the slower levels of memory hierarchy.

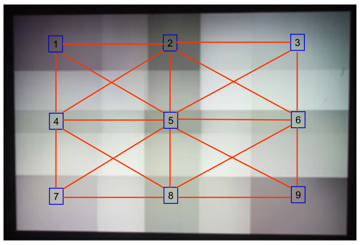


Fig. 3. The grid connection formed by the multi-display via spatial overlaps.

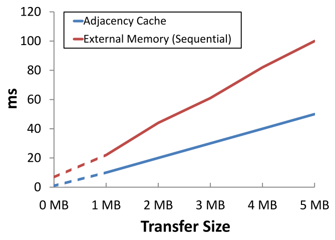


Fig. 4. Comparison between the transfer time for the AC and the EM, for different transfer size.

cache). For each SCD node, the collection of LC of adjacent SCD nodes, as given by the adjacency graph is termed adjacency cache (AC). The dashed red line in Fig. 2 represents a prioritized route (via the communication network) to get data between one compute node's LC and its neighboring's $LC(s)$. For each SCD node, its virtual external memory (VEM) is the combination of all the other SCD nodes' EM, not just those of the adjacent SCD nodes. The dashed black line in Fig. 2 represents the route that connects the EM of all the SCDs together (via the communication network).

Note that the GPU memory is not included in our memory hierarchy design. Usually, slower levels of memory hierarchy have larger capacity: for example, hard disk drives (HDDs) are slower than main memory, but have larger capacity. Since network data are fetched from the main memory, and data in the GPU, if required, has to be transferred via the main memory, fetching data from the AC is faster than fetching data from the GPU. In other words, transferring from the GPU memory to another machine is both slower, and has smaller capacity, than using the main memory. Thus, including the GPU in the memory hierarchy does not provide any advantage to our conceptual design. Further, the data that is held in the GPU is rendered by that machine. In all likelihood, that the data are not rendered at the same time by another machine. It may have been required in earlier, or may be required in subsequent frames, however, during those times, that data will not be in the GPU memory to be taken advantage of. Thus, including the GPU in the memory hierarchy does not provide any application-dependent advantages either.

3.2 Memory Performance

In this section, we analyze the performance of these memory structures: LC, AC, EM, and VEM, in order to establish a suitable memory hierarchy to determine the order in which data has to be requested from these memory structures. For each SCD, as expected, fetching data from its LC is the fastest, while fetching data from the VEM (EM of other SCD nodes) is the slowest. The data fetch speed comparison between the AC and the EM requires more discussion.

Fig. 4 shows the graph of time taken to transfer data of different sizes from the AC and the EM to the LC. The time to transfer data from the AC includes the network latency (a constant time overhead when getting the first byte, independent of the amount of data transferred), and the sustained transmission rate (typically, this is a linear function of the amount of data transferred). The y-intercept of the blue-line in Fig. 4 represents the network latency, and it is approximately 250 μs. The latency time can be isolated by measuring the time to send just one byte of data (including the packet header overhead) across the LAN. Similarly, the data transfer time from EM to LC includes disk seek time, and other operating system specific variables including disk-cache. This is again determined by the y-intercept of the red-line in Fig. 4, which is around 7 ms. For HDD, it is very difficult to isolate and measure this latency because at any given instance, the HDD is servicing multiple requests thus contaminating the measurement for a particular data request. Furthermore, there is also the scenario of read ahead where the HDD reads from surrounding blocks and caches the data. In summary, from Fig. 4 we see that the AC is faster than the EM, and hence the experiments and the empirical data show that the memory hierarchy from fastest to the slowest is LC, AC, EM, and finally, VEM.

HDD versus SSD. In the context of using HDDs as EM, we can also use SSDs as EM. Given that SSDs do not suffer from rotational latency and seek time, their number of Input/output operations per second (IOPS) is significantly higher that of their HDDs counterpart, but still orders of magnitude slower than main memory (LC). Therefore, it can be integrated into the memory hierarchy between main memory and hard disk. Hybrid drive is an example of such integration, where SSDs and HDDs are combined. A SSD could potentially replace a HDD as an EM, in which case the memory hierarchy, specifically the EM and the AC, will have to be restructured based on their relative performance. Currently, the latency for fetching data from the SSD-based-EM may almost be the same as the network latency of fetching data from the AC. From current technology trend, however, network bandwidth is faster than HDDs, and even SSDs (e.g. InfiniBand, 10 GigE). Therefore, if improvements in network bandwidth were considered, fetching large amount of data from the AC may be more efficient than fetching from SSD-based-EM, and the currently proposed memory hierarchy still holds. Nevertheless, replacing HDDs with SDDs will drastically improve the worst case scenario, where data has to be fetched from the VEM, which will be discussed in Section 4.

Although the proposed memory hierarchy may differ based on system configuration, as discussed above, once the hierarchy is fixed, our data distribution and management methodologies, which are detailed in the next section, will guarantee that any requested data will be efficiently delivered to the requesting node.

Our distributed shared memory architecture is independent of the application and the application data. Therefore, it can be integrated into any parallel rendering system. Currently, we use a grid topology to take advantage of the natural tiling of the displays. For a different topology, the optimal data distribution among the machines will change, but the overall concept of *LC*, *AC*, *EM*, and *VEM* will remain the same.

3.3 Data Retrieval

In an interactive model exploration and editing application, we expect users to continually interact with the data set. We expect them to navigate through it with different zoom levels, interspersed with interactive edit, delete, or create operations, on the scene's objects. In order to facilitate such applications, we must manage the flow of data to ensure that our system operates at an interactive rate.

We assume that the *VEM*, which is the union of all participating *EMs*, stores the entire scene. Since data editing is one of our primary objectives, we want to allow efficient updates during data modification. Unlike earlier work, which duplicates the data, we distribute the data to different *VEM*s, without duplication. Hence, we propose schemes for *partitioning* the complete data set in the object space and store the partitions in participating *EM*s on different machines. With such an object space partitioning, given a spatial region of interest, the *EM*, in which the data of that region resides, can be easily calculated.

During navigation, the entire display (or a smaller user-defined region) is considered as a single view frustum. This view frustum is divided among different SCDs based on the portion of the view frustum that the SCD's display occupies, with respect to the entire display. Only the data that is inside an SCD's view frustum is rendered by that SCD. Therefore, the data required by each SCD for its rendering will be brought from different *EMs*, including the SCD's own *EM*, and stored on the SCD's *LC*. In other words, the data stored on the *LC* is determined by the image space (view frustum based) partitioning, while the data stored on the *EM* is determined by the object space partitioning. Note that the partial view frustums will be slightly different depending on whether the tiled display is a multi-LCD or a multi-projector display. For LCDs, the view frustums will be partitioned across the different machines (Fig. 5), however, in a multi-projector display, although the data partitioning will be same, the partial view frustums will have overlaps with adjacent partial view frustums.

Whenever the global view frustum moves, each SCD node will recalculate its own view frustum. The SCD node will request the necessary data to render the scene from the data management system, based on the predefined memory hierarchy. The SCD node will first request the data from its *LC*. If the data are not available there, it will then try the *AC* (spatially adjacent SCDs' *LC*), which will most likely have the requested data in their individual *LC*, due to temporal coherence in the global view frustum during navigation. If still not successful, the SCD node will calculate the location of the required data in the *VEM*, which is unique since the data are partitioned and not duplicated. The SCD node will then fetch the required data from the appropriate SCD node's *EM*. In this sense, our proposed system is very similar to a single computer's memory hierarchy.

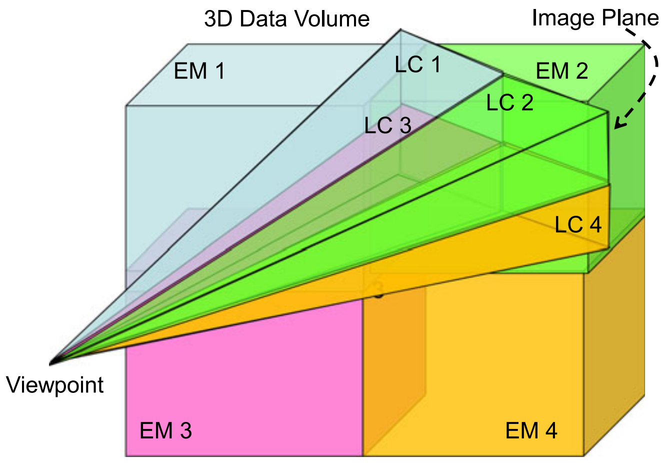


Fig. 5. An example of data partitioning and retrieval for a system composed of four SCD nodes: The 3D data volume is partitioned and stored on the *EM* of the four machines (colored differently in the figure). The content that an SCD node displays is dependent on the view point and view frustum, rather than what is stored on its local *EM*. The figure shows the view frustum rendered by each SCD. The data required to be rendered by an SCD has to be fetched from the *EM* and *VEM*, and stored on its *LC* for rendering. In the example shown, all the data to be rendered has to be fetched from EM2, and relevant portions are stored on individual *LCs* for rendering.

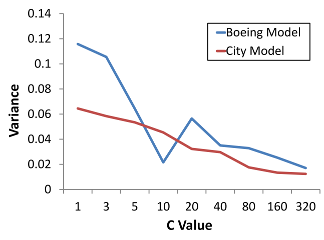
4 Data Partitioning and Distribution

Our distributed architecture eliminates the need for a central server, enabling each SCD node to operate independently from the other nodes. Hence, in the proposed approach, individual SCD's rendering load (how much data the node has to render on its display) no longer affects other SCD nodes, however, if any SCD node is under heavy load from servicing data (how much data the node has to send to other nodes), the slowdown will propagate to the other nodes. Hence, the challenge is to distribute the data set in a manner that will prevent any single SCD node from having all the data, for any particular view frustum. On the other hand, if the data are scattered across many SCD nodes, it cannot be efficiently retrieved. We seek a data partitioning method that would balance the data load of individual SCD nodes, while maintaining data coherency for efficient retrieval. This will minimize the network latency for fetching data, which in turn minimizes the delay between frames. In the following sections, we propose a data partitioning and distribution method that satisfies these competing constraints.

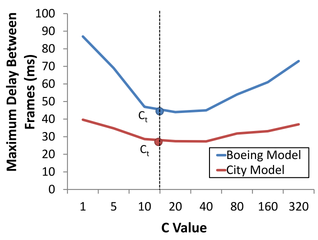
The data for the following sections is collected and measured on our prototype system of a 3×3 array of 9-projector display (Fig. 1), which is described in detail in Section 6. We use two different data sets for all of our results and analyses. To simulate scenarios where the data set is larger than the memory available, we restrict the main memory size to a fraction of the data set size. The first is a 4 GB City Model from University of California, Irvine, which has a relatively uniform spatial distribution. We restrict the main memory size to 500 MB for this model. The second is a 20 GB Boeing model, with a highly non-uniform spatial data distribution. We use a main memory size of 2 GB for this model.

4.1 Interleaved Data Partitioning

In this section we describe the modulo interleaving method, an interleaved data partitioning scheme for the 3D data, on



(a)



(b)

Fig. 6. (a) This plot shows how changing value of c (granularity of interleaving) changes the variance in the load across the SCDs. Variance stagnates beyond a certain c, indicating that no further improvement in load balancing can be achieved by increasing c. The single outlier to this trend is the lower variance for $c = 10$, for the Boeing data, whose highly non-uniform distribution is shown in Fig. 9. (b) This plot shows the maximum delay between frames, with changing value of c. We denote the optimal c, having the minimal maximum delay, as c_T.

a 2D grid, such as those commonly used in walkthrough environments. With minimal effort, this technique can also be extended to apply to a 3D grid as well. Let us consider the bounding box of the data, and let us designate one of the planes as the ground plane. This is relatively easy for massive walkthrough models, where the notion of the ground plane naturally exists. Let us consider that we have an array of $m \times n = N$ SCD nodes. The ground plane is divided into a $cm \times cn$ grid, where $c \geq 1$ is an integer that signifies the *granularity of interleaving*. As the value of c increases, it indicates a transition from a coarser, to a finer, interleaving. The data above each of the $cm \times cn$ grid constitutes a *cell*, the smallest chunk of data to be given to an SCD node. Each SCD node (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$, is assigned to a cell (i', j'), based on modulo functions $i' = km + i$ and $j' = kn + j$, where $0 \leq k \leq c - 1$. Fig. 5 shows an example for $m = n = 2$ and $c = 1$. Fig. 9 shows data distribution for $m = n = c = 3$. Data interleaving based on space filling curves [4], [30], which are used to characterize locality, leverages spatial coherence in only one dimension, whereas our proposed interleaving leverages spatial coherency in two dimensions.

The modulo interleaving has two main advantages: uniform data distribution and uniform network load distribution.

Uniform data load. A finer granularity of interleaving $(c > 1)$ allows for a more uniform distribution of data among EMs, even if the data are non-uniformly distributed in space. Let the amount of data at the EM of the SCD node (i, j) be $A(i, j)$. This is considered to be the load at the SCD node (i, j). Fig. 10 shows the $A(i, j)$ for different SCDs as a grayscale image for different values of c. The variance in $A(i, j)$ across all of the machines will measure how well-balanced the data distribution is, at the time of the calculation. Let us consider the Boeing model, a highly non-uniformly distributed data set. Fig. 6a shows that the variance decreases with increasing value of c, and stagnates beyond a certain c. Note that due to the varying local density of the data in the model, outliers can sometimes be seen, where their variance do not steadily decrease with increasing c, as shown in Fig. 6a. With such a high non-uniformity in the data distribution, other data balancing methods may be utilized, such as the scene adaptive data balancing method, as detailed in Section 4.2.

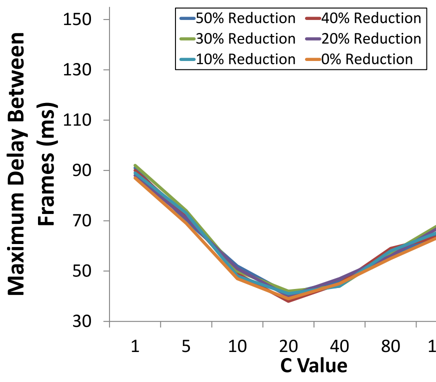


Fig. 7. Scalability of the proposed data distribution method. The performance of the system remains the same as the size of the data, the disk transfer time, and the network bandwidth are reduced by the same percentage. As the data size increases and the hardware improves, our data distribution scheme scales with the new platform by efficiently utilizing all the SCDs to avoid bottlenecks and to scale linearly.

Uniform network load. When an SCD needs spatially coherent data that are within its view frustum to be rendered, finer interleaving indicates that the data need to be fetched from multiple SCDs, thereby alleviating the network service load from any single SCD. Therefore, the total time to fetch the data and render the frame is reduced by increasing the value of c. Beyond a certain value of c, however, the granularity of the data is so fine that too many units of data are requested from each EM, which would require multiple disk seeks to each EM to fetch all of the required data. This overhead offsets the benefits of reduced network bottleneck. Fig. 6b illustrates this. The maximum latency to fetch the data across multiple navigation paths, when navigating through two very large models (Boeing and City), with the network bandwidth set to $1{,}000$ Megabits/sec, is shown in the figure. The value of c, at which the maximum latency is lowest, indicates the optimal granularity, and is denoted with c_T.

To demonstrate the scalability of this data partitioning scheme, as the network bandwidth, the size of the data, and the disk access time are decreased in the same proportion, the maximum latency is plotted against increasing value of c (Fig. 7). For example, when the network bandwidth, data size, and disk transfer bandwidth are reduced by 50 percent, the maximum latency is similar to when they were not throttled. The similarity of these curves indicates that our scheme scales linearly, in terms of data complexity and hardware, such that no single one SCD becomes a bottleneck. Note that c_T also remains relatively constant during these proportional changes. From Fig. 6b, which studies the data distribution for two models that vary greatly in size and spatial distribution, it is evident that c_T is more dependent on system parameters (like network bandwidth and HDD performance), than the data characteristics. Therefore, after a limited number of trial and error, an optimal c_T can be easily identified for any particular system.

4.1.1 Comparison to a Close-to-Optimal Greedy Approach

Optimal data load balancing across all SCD nodes is an NP-complete problem, as presented by Cormen et al. [6].

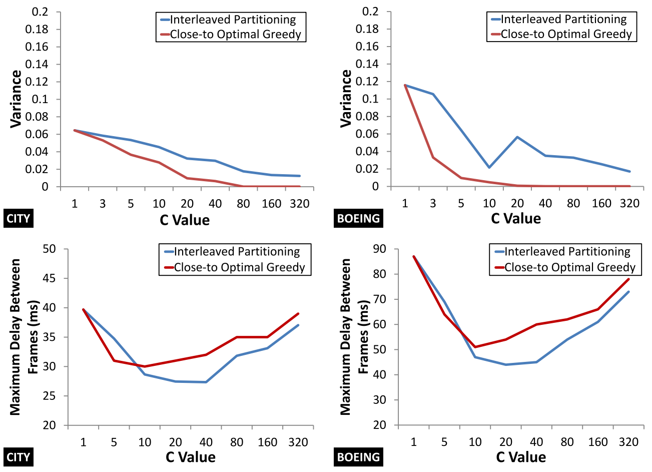


Fig. 8. This figure compares the variance (top) of the data distribution and the maximum delay between frames (bottom) of our interleaved data distribution (blue), with respect to a close-to-optimal greedy approach [12] (red)—for the City (left) and Boeing (right) model. Our interleaved distribution achieves a data load balancing close to this greedy approach. It also shows significantly lower maximum delay, benefiting from a spatially coherent layout.

It has been shown by Graham [12], however, that an approximate greedy approach exists, which is very close to optimal. In this method, the cells are first sorted from largest to smallest (in terms of the amount of data it contains). The largest cell is then iteratively assigned to the least loaded SCD node, as illustrated in Fig. 9. This approach is better than other existing approximate methods, however, it does not consider the spatial coherence of the data and can encounter network bottlenecks for some of the view frustums. Fig. 8 shows the difference in the variance of data across the machines for the greedy approach and for our data distribution approach. It shows that our method achieves a load balancing close to that of the greedy approach, however, our data distribution performs better in rendering performance, due to better preservation of spatial data coherence. Fig. 8 shows the maximum delay between consecutive frames for our approach and the greedy approach. For a small c, the greedy approach can still maintain some semblance of spatial coherence (Fig. 9) and hence, performs similar to interleaved partitioning. As c increases, however, especially near optimal c_T, the interleaved partitioning starts to exhibit significantly better performance.

4.1.2 Effect of Memory Hierarchy on Performance

The memory hierarchy has a significant effect on the performance of the system. It is unconventional, but we propose that the memory hierarchy should prefer the adjacency cache over the secondary storage, for many common system configurations. In the experiments that study the effect of the AC, we compare the maximum latency between two consecutive frames, for different values of c, under three different conditions: (a) the proposed memory hierarchy is used—LC, AC, and then EM or VEM (Fig. 11a), (b) if the data are not found in the LC, the data are directly fetched from the EM or the VEM, without checking the AC, and (c) the memory size of the AC is changed (Fig. 11c). The results of these experiments are shown in Fig. 11. The first two experiments [(a) and (b)] were performed with different network bandwidths, ranging between 500-1,000 Mbps. In contrast, the third experiment (c) was performed with different AC size, but the network bandwidth, data set size, and disk transfer rate, was kept constant.

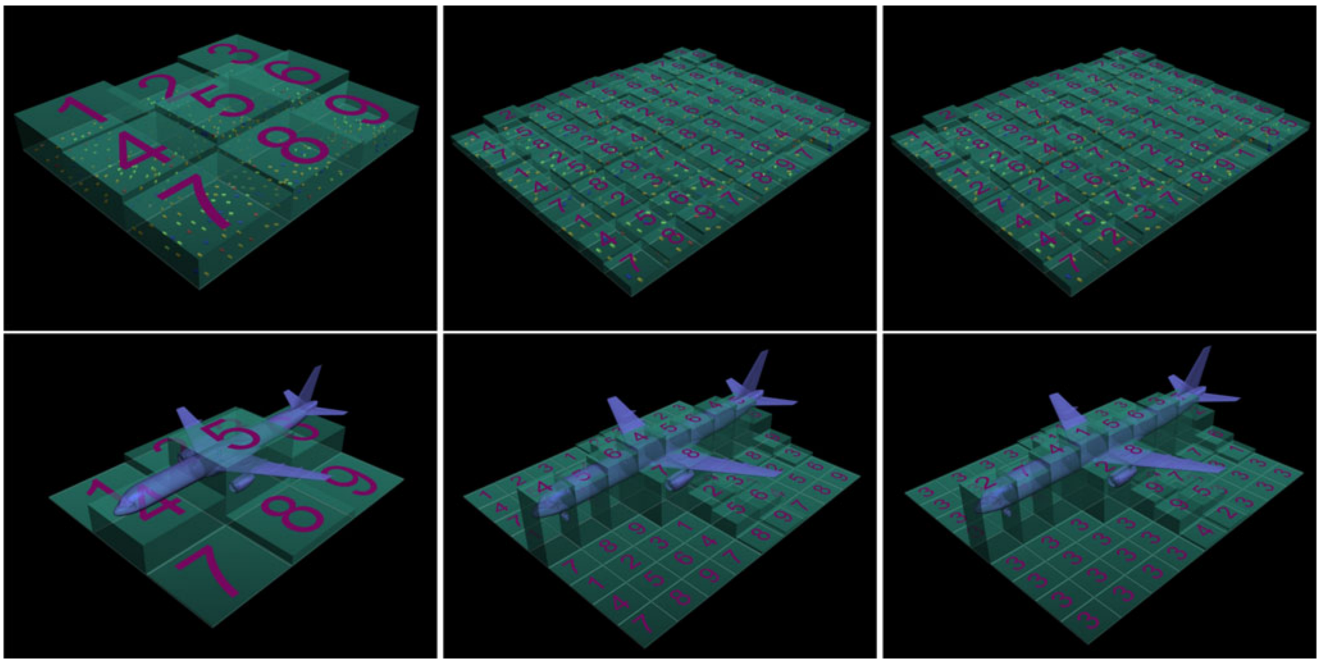
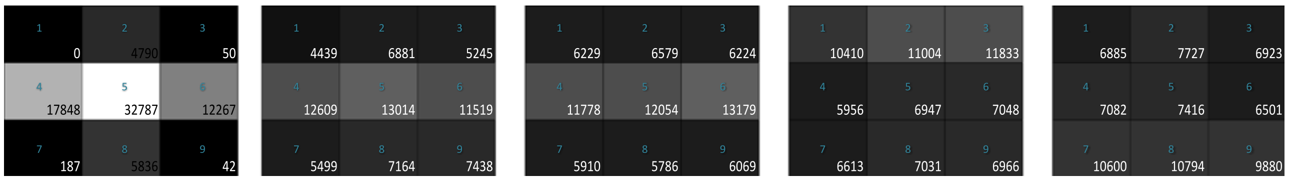


Fig. 9. This figure illustrates the interleaved data partitioning method on a 3×3 array of SCD nodes, for a 350 million triangles, 20 GB, Boeing 777 model (bottom), and a 4 GB, 110 million triangles, City model (top). The cells show the spatial partitioning. The height of the cell denotes the amount of data in that cell. The number in the cell is the SCD id, whose EM stores the data for that cell. The left column shows the data distribution without any interleaving (i.e. $c = 1, m = 3, n = 3$). The center column shows the interleaved data distribution for $c = 3$. The right column shows the data distribution using a greedy algorithm.



(a) $c = 1$ (b) $c = 5$ (c) $c = 20$ (d) $c = 40$ (e) $c = 160$

Fig. 10. This illustrates the load, $A(i, j)$, for different values of c across the SCDs, in a system with $m = 3, n = 3$, for the Boeing model. Brighter colored cells indicate a larger and positive difference in the data load. Note that with increasing *c*, the load balancing is better, and the data load at all SCDs converge to the average expected load.

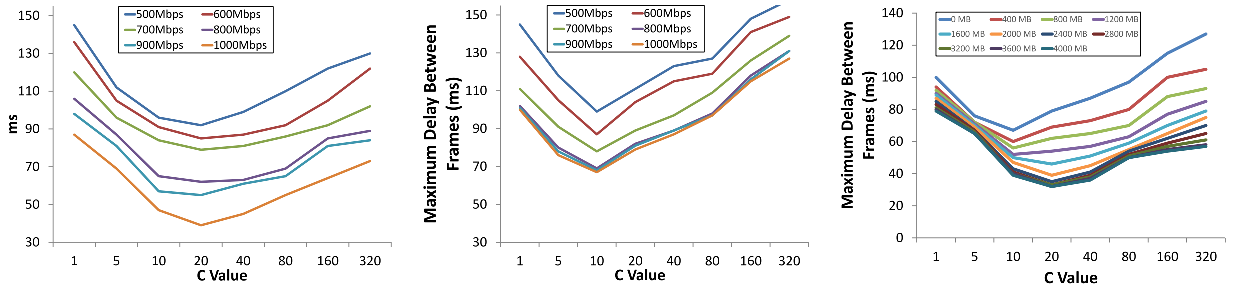
- Overall, Fig. 11a's maximum delay between frames is less than that in Fig. 11b's. This is because Fig. 11b depicts a scenario where there is a cache miss, and data need to be fetched, in worst-case, from the VEM, without checking the AC. In this scenario, each level of the memory hierarchy will need to be checked, except for the AC. In addition, the VEM is based on network bandwidth, and since this bandwidth is throttled for this experiment, its performance is suboptimal. These two conditions combined cause the total time spent requesting data on a slow network to be greatly extended. Therefore, because this problem is not observed in Fig. 11a, it can be concluded that the use of the AC, before the VEM, greatly reduces the maximum latency, thus, validating our proposed memory hierarchy.
- For c greater than c_T, the increase in delay between frames is more pronounced in Fig. 11b than in Fig. 11a. This substantiates our contention that the increase in latency is due to additional disk seeks in the EM, since when all data are accessed solely from the EM/VEM and not from the AC (Fig. 11b), this delay is more pronounced than in the scenario where the AC is used (Fig. 11a).
- Fig. 11b shows that when the AC is not used, the performance stagnates, even when the network bandwidth is increased. This is due to the fact that, for these scenarios, disk access time is the performance bottleneck. Therefore, increasing the network bandwidth will not improve the performance because the bandwidth will not be completely utilized. This also leads us to conclude that, if the network speed increases, the role of the AC will become even more critical since it will enable us to utilize the bandwidth to its full extent, as shown in Fig. 11a.
- Fig. 11c shows that the performance of the system does improve with larger AC, since it will result in fewer cache misses.

4.1.3 Effect of Data Load Variance on Performance

Our previously proposed interleaved data distribution scheme has a characteristic that impacts system performance: variance. This will be further discussed in the following sections.

To understand how load variance affects performance, we first need to understand why SSDs were chosen for the experiment. The increase in maximum delay between frames, when $c > c_T$, as shown in Fig. 6, was due to large number of disk seeks because of data fragmentation at high values of c. This was further indirectly verified by the pronounced increase in delay between the frames due to more disk access (seen in Fig. 11b when compared to Fig. 11a). A direct verification of this hypothesis is also needed. Therefore, an experiment is conducted using a device that has negligible (or constant) data seek time as the EM, instead of using HDDs. We chose SSDs for this purpose.

The result of replacing HDDs with SSDs is shown in Fig. 12 and this explains how SSDs performs differently than HDDs. The parameters for the experiment are the same as the ones used to generate Fig. 11c. Fig. 12 shows that the maximum delay monotonically decreases when SSDs were utilized, compared to the high delay shown in Fig. 11c, where HDDs were used. This suggests that the increase in maximum delay, for $c > c_T$, is primarily due to the data seek time in HDDs. Let us discuss the case when the size of the AC is zero. In this case, all data have to be



(a) (b) (c)

Fig. 11. Effect of AC on the rendering performance. All plots show the maximum latency between frames for different granularity of data distribution (values of *c*), different network bandwidth (a and b), and different size for the AC (c). The maximum transfer volume for one node per frame is 3 MB, without any throttling. (a) If the data are in the AC, then they are accessed from there; else, data will be retrieved from the VEM. (b) If the data are not in the LC, then they are directly fetched from the VEM. (c) Increasing the AC size does improve the performance, however, only up to a certain value.

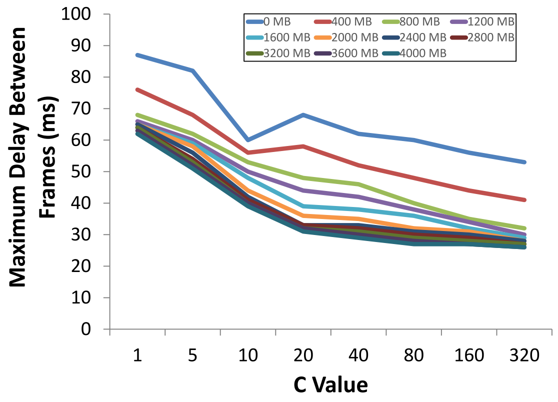


Fig. 12. This plot shows the maximum frame delay when an SSD is used, in place of a HDD, for different AC size. The network bandwidth, data set size, and disk transfer rate remain the same.

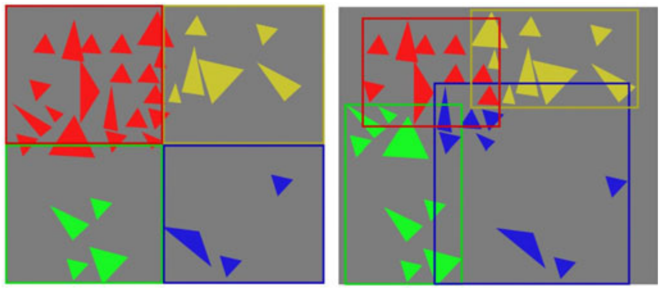


Fig. 13. In scene adaptive load balancing, cells are allowed to grow or shrink in a constrained manner (to retain most of the spatial coherence), while overlapping with their adjacent neighbors to achieve better data load balancing. The left shows four different colored cells (containing primitives of the same color), each assigned to a different machine. Note that the red one is the most loaded machine, while the blue one is the least loaded. The right figure shows the bounding boxes, and the triangle in each machine, after applying the scene adaptive balancing scheme. The bounding boxes have grown or shrunk and now are overlapping one another. Data are still partitioned across the machines without any duplication, but they are more uniformly distributed across machines than before.

accessed from EM/VEM since nothing can be cached locally in the LC, or in the AC, since its size is zero. The curve in Fig. 11c shows the behavior of HDDs for this scenario. Notice the difference between this and the curve in Fig. 12. The reason for this is because SSDs do not suffer from rotational latency and seek time, thus, the performance of the system is dependent on the maximum time taken to transfer the required data from VEM. This maximum transfer time is proportional to the maximum data serviced by an SCD node.

Referring back to Fig. 6a, which shows the variance of data (for the Boeing model) and the maximum data handled by any SCD node. Notice the uncanny similarity of this curve with the one shown in Fig. 12 for AC size equals zero. The reason for this is because when an SCD node requests data, it will fetch the data directly from the VEM. This process is dependent on how well-balanced the data are. Well-balanced data imply that more SCD nodes are available to service data to the requesting SCD node, resulting in a more uniformly distributed transfer load across multiple nodes, thus, improving performance. This clearly indicates that when the AC is zero, variance directly affects the system performance.

Fig. 12 also shows that when the AC is small, it cannot hold all the data that are rendered in the local SCD; therefore, even if the view frustum did not change, there will be cache misses, resulting in data transfer on the network. The maximum delay between frames for this scenario is inversely proportional with the AC size. After the size of the AC grows beyond the size required to hold the locally rendered data, however, only a view frustum change can trigger cache misses. This requires the SCD nodes to fetch the same amount of data across the network, irrespective of the size of the AC. Therefore, the maximum delay between frames is identical to each other. This is obviously shown in Fig. 12, where the curves form a cluster, when the AC size is greater than 2,000 MB. This indicates that when the AC is large, relative to the model size, variance has little impact on the system performance.

In conclusion, when the AC size is zero, the variance in data load distribution is an important parameter that dictates the system performance. While the variance is important, spatial coherency of the data assigned to the same SSD node is also important, as shown by Fig. 8. A modification of the interleaved data distribution algorithm, which will allow for further reduction of the variance, while maintaining spatial coherency, exists. Such algorithm will further improve the system performance, and is presented in the next section.

4.2 Variance Reducing Scene Adaptive Data Partitioning

For any given model, the proposed interleaved partitioning scheme does not take into account the spatial non-uniformity of model. The proposed scheme uses a fixed cell size and a cyclic interleaved assignment of cells to machines, resulting in a load balance that still deviates from the optimal solution, as shown in Fig. 8. In this section, we present a method to further balance the load by relaxing the size constraint of the cells by allowing them to overlap across the boundaries of the adjacent cells, as illustrated in Fig. 13. This allows the cells to grow and shrink, based on the density of the data in each cell, to achieve a better load distribution. We also constrain the extent of cell shrinkage and growth, to maintain a high spatial coherence and still have cyclic interleaved assignment. Further, although we allow cell boundaries to change, we do not allow for data duplication, to maintain a strict data partition.

Fig. 13, left, illustrates the initial bounding boxes. The data contained in these four bounding boxes can be mapped to a 2×2 cell of SCD nodes. The amount of data contained in these boxes need not be the same. After performing our proposed method, the bounding box of each cell would either grow or shrink to reflect the distributed data. The new bounding boxes, as shown in Fig. 13 (right), will either contain more or fewer primitives than before, and when these data are mapped to 2×2 SCD nodes, we achieve better data balancing across nodes. Fig. 14, illustrates the entire algorithm that has four iterations of the above processing shown in Fig. 13, but on a 3×3 array of SCD nodes.

Let the interleaved-partitioning data, that was generated from the method shown in Section 4.1, be the input into the this proposed data partitioning scheme. Our iterative data balancing procedure processes as many non-overlapping

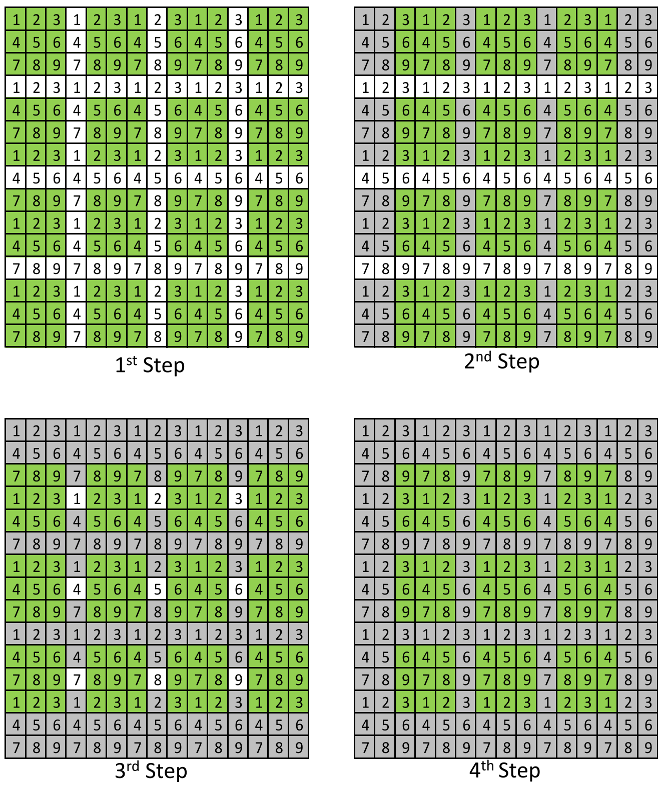


Fig. 14. A 3×3 array of machines, with interleaving of $c = 5$, resulting in 15×15 cells. This figure shows the different metacells processed, in parallel, for each step of the scene-adaptive balancing scheme. The cells not covered by a metacell in any of the previous steps is shown in white. Note that in four steps, all cells are processed.

3×3 grids of cells (called *metacells*) in parallel, as possible. These non-overlapping metacells are spaced to be at least one row and one column apart, as shown in Fig. 14. Each iteration of our load balancing procedure has four steps. The metacells chosen in each step will overlap with two metacells processed in the previous step. This constraint ensures that all of the metacells will be covered in the four steps of each iteration. The processing that is done in each step is described below.

Let each cell in a metacell be denoted by p_i, $1 \leq i \leq 9$, and the amount of data be $A(p_i)$. The goal is to reduce the variance for the load (the $A(p_i)$) of the cells in each metacell, in order to make the load distribution more uniform. We define a scale factor f, which is a factor up to which an underloaded cell can expand its bounding box to include more primitives. Let M denote the average load of all cells in the metacell. Let p_m be the cell with the maximum load among the cells in the L_c. The amount of extra load p_m has can be calculated using the formula $E_m = |A(p_m) - M|$. Our greedy approach to load balancing allows the bounding box of the primitives in each cell to grow or shrink, to include or exclude primitives from and to its neighboring cells, within the scale factor. The process starts with each cell having a list of cells, L_c, which are the cells currently involved in the load balancing process. Initially, this list includes all nine cells. A cell will take itself out of the process when there is no neighbor capable of sharing its load, or when the cell has grown or shrunk by f. The process ends when this list is empty.

Firstly, every cell communicates its load to all of the other cells in the metacell. If p_m does not have any neighbor whose load is less than M, then it does not have a neighbor capable of sharing its load. In that case, p_m broadcasts a message to all of the other cells and removes itself from the L_c. If p_m has a non-zero number of neighbors with load less than M, it starts sharing its load with its neighbors. Starting with the least loaded neighbor p_l, it shares the load by expanding and shrinking the bounding boxes of p_l and p_m, respectively, along their boundary. It can only expand or shrink its bounding box up to a factor of f, or when the amount of data transferred is at most $E_l = |M - A(p_l)|$, or E_m, whichever happens first. Following this, the new load of p_m and p_l are communicated to all the other cells in L_c. Next, p_m recalculates the E_m to find the remaining load to be shared. The process then moves to the next underloaded neighbor of p_m for further load sharing. This process continues until either $A_m = M$, until the process runs out of neighbors to share the load with, or until all of p_m's neighbors' bounding box have grown to the maximum possible extent. When this process stops for p_m, it communicates to all of the other nodes and takes itself out of the L_c. The process then moves to the cell with the next highest A in the L_c. This continues until L_c is empty. Note that since the data only travel from more loaded cells to less loaded cells, and the process converges as the load of each cell reaches M, no cell can ever shrink to zero. The pseudocode of this scene adaptive data distribution scheme is detailed in Algorithm 1.

Algorithm 1. Scene Adaptive Data Balancing

```
$StepCount \leftarrow 4$
$BLK \leftarrow$ Blocks inside grid
$D \leftarrow$ Maximum distance between cell
$NC \leftarrow$ Number of cells in block
for all $StepCount$ do
  for all $BLK$ do
    Sort cell by data size
    $T \leftarrow$ total data size of all cells
    $AVGSIZE \leftarrow T/NC$
    $DELTA \leftarrow$ A non-negative number (~5% of the AVGSIZE)
    while not All cells visited do
      $C \leftarrow$ cell with the least amount of data
      $OC \leftarrow$ all other cells with more data than $C$
      $CLOAD \leftarrow$ data size of $C$ below AVGSIZE—DELTA
      if CLOAD $< 0$ then
        for all OC do
          if OC's distance from C $<$ D then
            $OCLOAD \leftarrow$ data size of $OC$ above AVGSIZE $+$ DELTA
            if OCLOAD $> 0$ then
              $MIN \leftarrow$ minimum of $|(CLOAD, OCLOAD)|$
              Re-allocate MIN amount of data from OC to C
            end if
          end if
        end for
      end if
      Remove $C$ from list of cell to visit
    end while
  end for
end for
```

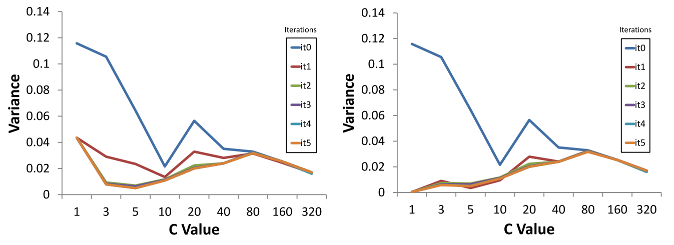


Fig. 15. The variance in the data load is reduced over multiple iterations of the scene-adaptive balancing scheme. This figure shows the change in the variance for different values of c, and for the bounding-box scale constraint of $f = 0.5$ (left), and $f = 1$ (right), for the Boeing Model. Note that as f increases, relaxing the spatial coherence constraint, the variance in the load is further reduced.

Figs. 15, 16, and 17 presents the results for this scene adaptive data distribution scheme. Fig. 15 shows the variance of the data distribution, after each iteration, for different values of c. Each iteration progressively reduces the variance, and it converges in just two iterations. Note that as the spatial coherence constraint is reduced with a higher f, the data load variance is reduced even further. Fig. 16 shows that data balancing using the scene adaptive data management is always better than the one achieved by interleaved data distribution. For any given f, note that as the cell size decreases (c increases), the bounding box gets smaller, and the amount of data change allowed in the bounding box also reduces. For models with non-uniform distribution of data, such as the Boeing model, this small change in the bounding box is not enough to balance the data load. Therefore, as the value of c increases, the variance in the data load also increases, especially for models with non-uniform spatial distribution of data.

In terms of actual system performance of the walk-through rendering application, Fig. 17 shows that the scene adaptive data balancing scheme's maximum delay between frames is lower than that of the fixed interleaved partitioning scheme, for different values of f. First, note that the maximum delay between frames after scene adaptive balancing is always lower than that from the interleaved data distribution, irrespective of the value of f. Next, similar to the preferred value for c for lowest delay between frames, this graph shows a preferred value for f. For example, for Boeing model, the maximum delay is minimal for $f = 1$. For $f = 0.5$, the data load is not well-balanced. For $f = 2.0$, the spatial coherence of the data is not adequately maintained. $f = 1$ provides the preferred combination of a balanced load and spatial coherence, yielding the best performance.

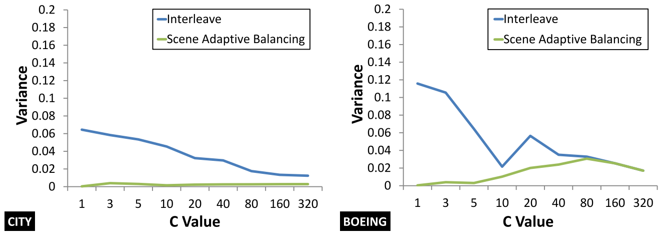


Fig. 16. The variance of the data load, resulting from the scene adaptive data balancing scheme (in bolded green lines), is much smaller than the interleaved approach (in bolded blue line), both for the City (left) and the Boeing (right) model. Note that, by design, the variance for the scene adaptive balancing scheme cannot be worse than the interleaved approach.

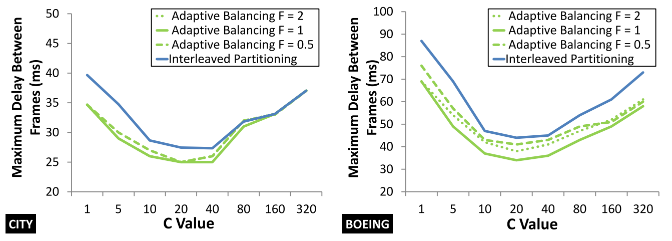


Fig. 17. This figure shows the effect of the scene adaptive data balancing scheme for different f ($f = 0.5, 1.0$, and 2.0), compared to the maximum delay between frames, for the City (left), and Boeing model (right).

The effect of f is much more dramatic for the Boeing model (with non-uniform data distribution) than for the City model (with almost uniform data distribution). For the City model, the initial data distribution is almost uniform, hence the effect of the scene adaptive data balancing yield similar results for each f. Note that $f = 2$ overlaps with $f = 1$ for $c = 1$ to $c = 20$. From this we find that there exist a sweet spot in f, which balances spatial coherence with similarity in load, to achieve optimal performance. Finally, as expected, the effect of changing f on the system performance is much more dramatic for a non-uniformly distributed model (the Boeing model) than for a uniformly distributed model (the City model), due to the much lower variance in the data distribution with a larger f.

Fig. 18 shows the gray scale visualization of the amount of data in each cell, for different iteration count, when the scene adaptive data balancing scheme is applied on the Boeing model. It is evident from the decreasing contrast of the visualization that the load is iteratively more well-balanced. This is shown for two different values of f. With a higher f, the load is better balanced, but has less spatial coherence. Since there is no data duplication, f affects the spatial coherency of the data by controlling the spatial range in which the data can be moved from its original interleave partition. For example, in Fig. 17, a value of $f = 2.0$ would allow the data block to move up to a maximum of two unit blocks away from its original interleave partition.

5 Navigation and Modification

Based on the results of the scene adaptive data load balancing scheme, data are distributed to individual SCD nodes and stored in their EM. During the virtual navigation through the scene, the view frustum is changed based on the user's input and this view frustum is broadcast to all SCD nodes. Each SCD node would compute the part of the view frustum it is responsible for, as shown in Fig. 5, and fetch the required data for this partial view frustum from the fastest level of the memory hierarchy, where the data are present.

After the initial data distribution is finished, a meta file is used to locate the required data. A look-up table is generated that consists of the object ID, the object's bounding box, all of the page IDs that contain the object data, and the SCD node's ID, for which the pages reside in (for the VEM). In addition to the look-up table, the meta file also contains the geometry of the interleaved grid that was used to distribute the data. At run time, if an SCD node's view frustum intersects a grid cell (bucket), the SCD node will fetch all

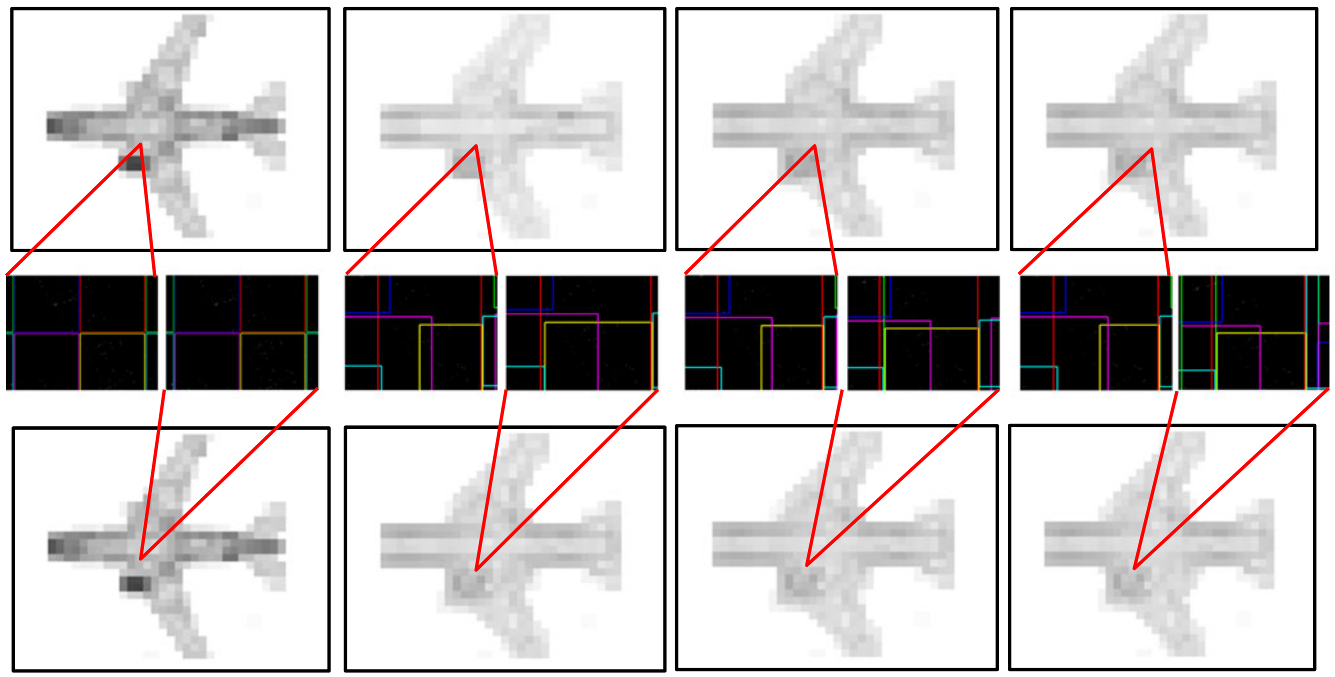


Fig. 18. This figure is a pictorial representation of the amount of data in each cell for $c = 5$, for a 3×3 system, where the amount of data is represented by a gray scale value. From left to right, we show the results after 0, 1, 5 and 23 iterations, respectively. Since the variation in the data size across different cells is large, we use a log scale conversion to gray values between 0 and 1. The top row shows this result for $f = 0.5$, and the bottom for $f = 1$. Note that for the former, the shrinking and growth in the bounding boxes are much less than the latter. The zoomed-in views show the data partition in the same small region of the scene, at the junction of the two red lines, and illustrate the growth and shrinkage of the bounding boxes, which are allowed to overlap, to achieve this load balancing.

data pages associated with that grid cell and store it in its LC for rendering. If the amount of data that needs to be fetched exceeds the LC, out-of-core algorithms are used, which will negatively impact the rendering performance. Information about the object's bounding box may be used to compute the image-space projection-error, in order to choose the appropriate object level of detail (LOD) that has to be rendered.

Interactive editing applications expect users to continuously modify and interact with the model. In addition to operations such as addition and deletion of objects, our proposed system supports many edit operations such as scaling, translation, rotation, and shearing. Such modifications are assumed to occur at the object level, which are well supported by a page-based data layout [23]. When the user performs a delete operation on the selected data, data are removed from all the nodes that have this data. Similarly, during an insert operation, new data are inserted, in an interleaved manner, based on their object space location, to all of the relevant nodes. Note that only the objects that are displayed can be edited. So by design, if an object is selected for editing, all page blocks associated with that object would be in the memory (LC) of SCD nodes that are displaying the object, or being streamed using out-of-core techniques. Any edit operation done will be updated across all SCDs. Once the object has been deselected, the SCD node that currently renders the edited object (and the one with smallest id, if there are multiple nodes rendering the object), would commit the change to the VEM, and a message is sent to all SCD to invalidate the object in their LC. The updated object must then be fetched from the VEM. Transformation parameters are sent to all SCD nodes rendering the object. With every insertion or deletion, the data can change considerably—both in terms of their size and their spatial organization. Therefore, through interleaved distribution, data are inserted to, or deleted from, the data management system first, followed by the scene adaptive data balancing. The distribution achieved by using this approach is almost identical to the one achieved for modified data detected during system startup.

As a system design principle, the scene adaptive data balancing step is performed during the idle cycles, to minimize the impact on the users' interaction. To make the process more efficient, the re-balancing step is run only on the cells which belong to the metacells included in, or adjacent to, the metacells in the view frustum (Fig. 19). Since the constraints of the growth and shrinkage of the bounding box keep the data local, and since it is reasonable to assume that data edits primarily occur within the view frustum, having this restriction results in a fast re-balancing.

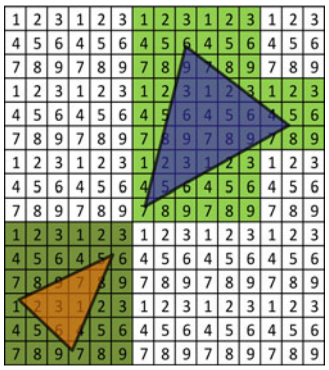


Fig. 19. The cells affected by the scene adaptive balancing scheme, following a data modification for two differently sized view frusta.

Our proposed system treats any data modification (e.g. translation, scaling) as a deletion of the existing object, followed by an insertion of the modified object; however, these edit operations may take multiple frames to balance. For example, for scaling, the user has to select the object and scale it to the desired size, over multiple frames. Similarly, for translation, the user has to select the object and move it to a different location, over multiple frames. Instead of committing multiple deletions and insertions for each frame, upon the object selection, the pages in each EM are marked. When the user is performing the operation, each SCD node will render the modified data using the transformation parameters. Once the user is done with the modification, indicated by releasing the object, the marked pages are deleted, followed by an insertion of the modified object. This strategy has two advantages. First, the user continues to get interactive feedback of his modifications while he is performing the edit. Second, the network and EMs are not taxed with excessive data movements, only those necessary for the modification to be committed.

6 Implementation and Results

Firstly, we want to stress that our work is a distributed data load balancing and management technique. Although our implemented system includes parallel rendering aspects, the proposed method is not a rendering load balancing technique.

System infrastructure. Our distributed interactive navigation and modification paradigm has been implemented on a 3×3 grid of SCD nodes, projected from a 3×3 multi-projector display, in our lab (Fig. 1). Each SCD node consists of a $1{,}024 \times 768$ Epson and Canon presentation projector, of 3,500 lumens brightness, and an Intel i5 3.3 GHz PC. Together, they form a $8' \times 6'$ display. All of these PCs are connected over a gigabit LAN. Each PC is equipped with an nVidia GeForce 560 GPU (1 GB), and 8 GB RAM. In order to test our architecture and associated methodologies, and in order to simulate the effects of having significantly smaller main memory than the size of the model, the main memory was limited, based on the size of the model. For example, the PCs were limited to 500 MB RAM per PC for the 4 GB City model, and 2 GB RAM per PC for the 20 GB Boeing model. In the prototype, textures were not considered; however, our architecture does not have any inherent limitation that would prevent it from being extended to textured models. For the registration of geometry and color, existing techniques, presented by Bhasker et al. and Sajadi et al. [2], [25], were used.

The application is written in C++ using OpenGL graphics library. All prioritized communication route were implemented in software. Out-of-core rendering for the GPU was utilized to stream the updated data to the GPU. All the data primitives are triangles. For editing tasks, all objects that can be edited are identified by an ID, and edit operations are performed on the object, using the object's ID.

Data layout on EM. The secondary storage uses the page-based data structure data layout proposed by Sajadi et al. [23] to store the data in a cache-coherent manner, on the disk. In this approach, the geometric data stored in a disk-page (of size determined by the operating system) is self-contained. For example, for each triangle stored in a page, all of its vertex information are also stored in the same page.

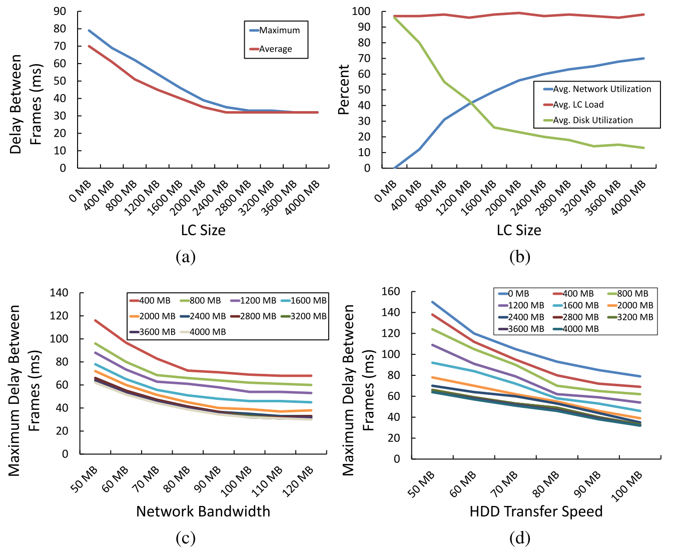


Fig. 20. System performance when c is set to the optimal value of 20 and f is set to 1. (a). Maximum and average delay between frames when system parameters are at their default values. (b). Resource utilization. (c). System performance with different network bandwidths. (d). System performance with different HDD transfer speeds.

The group of triangles stored in a page are all related: for example, they may belong to the same LOD of the same object, or they may be spatially close and hence may belong to the same cluster. A disk-page hierarchy is built by clustering the primitives of the scene in terms of their spatial proximity, and laying them linearly on the disk in multiple pages. A K-d tree is generated for the bounding boxes of the primitives for each disk page. The leaf nodes of the K-d tree have pointers to the disk-pages, and not individual triangles. This K-d tree is stored as a metadata in each SCD node, and is used by the data-access sub-system middleware in order to easily locate, and fetch, the required object data in the VEM. The rendering system, when it has to fetch data from the VEM, will interface with the data access subsystem by requesting specific pages of data. A page-based data layout utilizes the properties of the physical disk to efficiently read and write data between main memory and secondary storage. The page size used is equal to that of the hard disk's block size, based on how the hard disk is formatted. Thus, each page can be read in its entirety from a single hard disk block. If the page size is not aligned with the hard disk's block size, a page may reside in multiple blocks, thus, introducing more I/O overhead.

Performance analysis. Using the optimal value of $c(20)$ along with the bounding box scale constraint $f(1)$, we performed a series of test to determine how the system parameters affect the overall system performance. The system parameters are: LC, network bandwidth, and hard disk speed. Fig. 20 shows how the LC size and the network bandwidth can affect the maximum delay between frames. Graph (a) shows the effect of the LC size on the system performance. Graph (b) overlays the average LC load with the average network utilization and average hard disk utilization. Note how the average LC load remains constant through out all LC size. This is due to the fact that he LC size is less than the data set size, therefore, the SCD node would try to cache as much data as its LC can hold. Graph (c) measures the impact of the network bandwidth on the system. When the network bandwidth is limited to 50 MB per second, the delay between frame is almost double the typical value in the previous graph for each *LC* size. Therefore, the network bandwidth is crucial to the overall system performance. The hard disk transfer speed graph also shows a similar behavior.

Frame synchronization. We use a simple distributed algorithm to achieve frame synchronization across multiple SCD nodes. One SCD node acts as the main controller and broadcasts a series of timed messages to all other nodes. The other nodes will measure and compensate their delay from the main controller upon receipt of these messages. This achieves a synchronization up to at most one frame (i.e. 32 milliseconds) difference, considering standard projector refresh rate. According to Shneiderman [28], a system response, to any user interaction, within 80 milliseconds is perceived to be instantaneous by humans. Our system can respond to any user input within 32 milliseconds. It may take up to an additional 32 milliseconds for the frames to be synchronized. Therefore, the maximum response time for the system is 64 milliseconds, which is well within Shneiderman's tolerance level; hence, it is imperceivable. The rendering system can operate in two synchronization modes: blocking and non-blocking. In blocking mode, all SCD nodes will announce when they are ready to swap the framebuffer. Once all SCD nodes have acknowledged that they are ready to swap their framebuffer, they will all perform the operation at the next synchronized time. In non-blocking mode, each SCD node will swap its framebuffer, independent of the other SCD nodes. After this action, it will wait for the next synchronized time. If a better synchronization method is desired, one can use an expensive hardware based solution, like the Genlock (as in ATI FirePro S400) or G-Sync (Nvidia). G-Sync makes it possible for the display refresh to be synchronized with the GPU render rate, thereby solving the problem of synchronizing the system display output with the display's screen refresh rate.

Model processing. Our method is specifically geared towards massive models. Therefore, it holds significant benefits for very large environments. Using our nine SCD nodes system, the Boeing model can be navigated, and objects can be interactively modified, at an average frame rate of 45 fps, even when each SCD's RAM is limited to 2 GB. On any given SCD node that has 4 GB of usable RAM, the entire City model can be loaded into that machine's RAM, however, the best frame rate achieved is 5 fps. This indicates that the main bottleneck is due to the rendering of the model. When using the model on our nine SCD nodes, since the model is much smaller than that of the Boeing's, the RAM is limited to 500 MB, but it can be navigated and modified at an average frame rate of 57 fps. This clearly demonstrates the advantages of our distributed techniques. The one-time pre-processing, including the page based data layout (using the technique proposed by Sajadi et al. [23]), the interleaved data partitioning, the pre-edit model adaptive data balancing, and the movement of the data to different SCD nodes, takes around 150 minutes for the Boeing model, and 45 minutes for City model. A 4K page size was used for the page based layout. Interactive rendering rates are still maintained, even after dynamic balancing is performed following any edits. Our implementation does not use levels-of-detail (LODs), however, the distributed data management technique does not depend on the use of LODs. Each LOD of the entire scene can be distributed independently to all SCD nodes using the same method described in the paper. It is important to note that our method distributes the data spatially; therefore, all the LODs of an object that occupies the same space, will be stored within the same cell. The rendering node can decide on the appropriate LOD to render, and fetch the required data from the nearest storage location in our memory hierarchy. Consistency of the choice of the appropriate LOD, for the same model, for any given viewpoint, across different SCD nodes, is guaranteed because the same code that is running on each SCD node is the same code that is running in our distributed data management and rendering platform.

TABLE 1
This Table Shows the Compression Size and Decompression Time, for Different Page Size

Page Size	Compressed Size	Decompression Time
4 K	1.8 K	470 μs
8 K	3.61 K	811 μs
16 K	7.18 K	1,411 μs
32 K	14.3 K	2,296 μs
64 K	28.5 K	4,351 μs

Rendering versus data management. Most existing systems focus more on the parallel rendering, leaving the data management to the application developer. Our distributed data layout and management system is complementary to all prior parallel rendering method. In other words, any parallel rendering system can use our distributed data layout and management system. Fig. 17 shows the results of one simple parallel rendering system, in which our system achieved an average frame rate of 30 fps for 350 million triangles. In comparison, Equalizer gets around 2.5 fps for a 225 million triangle model (refer to [7, Fig. 24], although it should be noted that the machines and GPUs used in that work is older and slower). We believe that the performance of parallel rendering systems, such as Equalizer, can be improved, by using our data management system.

Compression. It may be argued that data compression can be used to reduce the storage requirements, which will reduce the need for distributed storage, network data transfer time, and even out-of-core processing and rendering. While compression does reduce storage requirement, decompression of data for each page, however, will increase the latency and becomes a bottleneck for real-time rendering and interaction. As illustrated in Table 1, the decompression time for a small block of data—even when using a bitstream Huffman encoding, best known for its fast decompression—is so large that it prohibits real time rendering. If a transfer volume of 3 MB were fetched from the *VEM* for a particular frame, using 4 K page size, it would take approximately 352 milliseconds to decompress. On a gigabit LAN, 3 MB take approximately 30 milliseconds to transmit. If the data were compressed by a factor of 2, it would take approximately 15 milliseconds to transmit the data over the LAN, and an additional 352 milliseconds to decompress the data. Therefore, the gain in reduced data transfer time is overshadowed by the overwhelming cost to decompress the data page.

7 Conclusions and Future work

In summary, we have presented a distributed architecture for data management, for interactive navigation and modeling of large environments. The first main contribution of our work is the proposed distributed memory hierarchy that includes the local main memory (local cache—*LC*), the main memory of the adjacent nodes in the tiled display (adjacency cache), the local external memory and the external memory of all the other nodes in the distributed system (virtual external memory). The second contribution is the interleaved data partitioning and distribution method that reduces variance in the data load, while maintaining the spatial coherency scene data, stored in the same storage node.

We are in an age of data explosion. Soon, our virtual environments will be much bigger than the Boeing model (20 GB) used in this paper. Our system represents a versatile framework which, in the future, can be extended in many ways. A previous work by Roman et al. [21] presents a distributed interaction paradigm to interact with 2D data on a cluster of compute-display-interact (CDI) nodes. We are exploring the extension of this work to 3D models by modifying the front end of the user interface. This can lead to a system that offers extremely lucid and natural gestures for implementing the navigation and edits of data models. Since our architecture inherently can scale to multiple users, it facilitates co-located collaborative navigation and edits of large VR environment. Further, although our storage load balancing is complementary to rendering load balancing, we want to explore merging earlier work on rendering load balancing with our architecture. Future work includes extending our 2D data partitioning scheme into a 3D scheme and handling of dynamic data. We will have to address challenges such as how a 3D scheme would introduce additional data load imbalance since each SCD would now have many more neighboring SCDs. In general, highly dynamic data sets that change continuously with time (e.g. particle system or flow visualization) is a challenge for distributed data system. Unlike modeling applications, however, simulation systems come with analytical and predictive data movements. It will be interesting to explore how this can be leveraged in the context of our distributed data management scheme.

Acknowledgments

The authors would like to thank David Kasik from Boeing for helping obtain the 20 GB Boeing model. It has been a wonderful test model to experiment with our architecture. They would like to thank Prof. Sandy Irani of the University of California, Irvine for helping us with the theoretical analysis on proximity of optimality achieved by our interleaved data partitioning method. Duy-Quoc Lai is the corresponding author.

References

[1] W. V. Baxter III, A. Sud, N. K. Govindaraju, and D. Manocha, “Gigawalk: Interactive walkthrough of complex environments,” in *Proc. 13th Eurographics Workshop Rendering*, 2002, pp. 203–214.

[2] E. S. Bhasker, P. Sinha, and A. Majumder, “Asynchronous distributed calibration for scalable and reconfigurable multi-projector displays,” *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1101–1108, Sep. 2006.

[3] I. Boada, I. Navazo, and R. Scopigno, “Multiresolution volume visualization with a texture-based octree,” *Vis. Comput.*, vol. 17, no. 3, pp. 185–197, 2001.

[4] P. Campbell, K. Devine, J. Flaherty, L. Gervasio, and J. Teresco, “Dynamic octree load balancing using space-filling curves,” Williams College Dept. Comput. Sci., Williamstown, MA, USA, Tech. Rep. CS-03-01, 2003.

[5] M. Chapman and G. Heiser, “vNUMA: A virtual shared-memory multiprocessor,” in *Proc. Conf. USENIX Annu. Tech. Conf.*, 2009, pp. 2–2.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.

[7] S. Eilemann, M. Makhinya, and R. Pajarola, “Equalizer: A scalable parallel rendering framework,” *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 3, pp. 436–452, May/Jun. 2009.

[8] F. Erol, S. Eilemann, and R. Pajarola, “Cross-segment load balancing in parallel rendering,” in *Proc. 11th Eurographics Symp. Parallel Graph. Vis.*, 2011, pp. 41–50.

[9] S. E. Yoon and D. Manocha, “Cache-oblivious layouts of bounding volume hierarchies,” *EuroGraphics*, vol. 25, no. 3, 2006.

[10] B. Fitzpatrick. (2010). Memcached, Computer Program [Online]. Available: http://www. memcached. org

[11] E. Gobbetti and F. Marton, “Far voxels: A multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms,” *ACM Trans. Graph.*, vol. 24, no. 3, pp. 878–885, 2005.

[12] R. Graham, “Bounds on multiprocessor timing anomalies,” *SIAM J. Appl. Math.*, vol. 17, pp. 263–269, 1969.

[13] M. Guthe, P. Borodin, and R. Klein, “Real-time out-of-core rendering,” *Int. J. Image Graph.*, 2005.

[14] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, “Chromium: A stream-processing framework for interactive rendering on clusters,” vol. 21, no. 3, pp. 693–702, 2002.

[15] E. LaMar, B. Hamann, and K. Joy, “Multiresolution techniques for interactive texture-based volume visualization,” in *Proc. Conf. Visualization: Celebrating 10 Years*, 1999, pp. 355–361.

[16] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, “A sorting classification of parallel rendering,” *IEEE Comput. Graph. Appl.*, vol. 14, no. 4, pp. 23–32, Jul. 1994.

[17] B. Moloney, M. Ament, D. Weiskopf, and T. Moller, “Sort-first parallel volume rendering,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 8, pp. 1164–1177, Aug. 2011.

[18] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, “Global arrays: A nonuniform memory access programming model for high-performance computers,” *J. Supercomput.*, vol. 10, no. 2, pp. 169–189, 1996.

[19] D. Reiners, “Opensg,” Ph.D. dissertation, Department for Virtual and Augmented Reality, TU Darmstadt, Darmstadt, Germany, 2002.

[20] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, “Sage: The scalable adaptive graphics environment,” vol. 9, no. 23, pp. 2004–09, 2004.

[21] P. Roman, M. Lazarov, and A. Majumder, “A scalable distributed paradigm for multi-user interaction with tiled rear projection display walls,” *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 6, pp. 1623–1632, Nov./Dec. 2010.

[22] M. Roth, P. Riess, and D. Reiners, “Load balancing on cluster-based multi projector display systems,” in *Proc. 14th Int. Conf. Central Eur. Comput. Graph., Vis. Comput. Vis.*, 2006, pp. 55–62.

[23] B. Sajadi, Y. Huang, P. Diaz-Gutierrez, S. Yoon, and M. Gopi, “A novel page-based data structure for interactive walkthroughs,” in *Proc. Symp. Interactive 3D Graph. Games*, 2009, pp. 23–29.

[24] B. Sajadi, S. Jiang, M. Gopi, J.-P. Heo, and S.-E. Yoon, “Data management for SSDs for large-scale interactive graphics applications,” in *Proc. Symp. Interactive 3D Graphics Games*, 2011, pp. 175–182.

[25] B. Sajadi, M. Lazarov, M. Gopi, and A. Majumder, “Color seamlessness in multi-projector displays using constrained gamut morphing,” *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1317–1326, Nov./Dec. 2009.

[26] B. Sajadi and A. Majumder, “Markerless view-independent registration of multiple distorted projectors on extruded surfaces using an uncalibrated camera,” *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 6, pp. 1307–1316, Nov./Dec. 2009.

[27] B. Sajadi and A. Majumder, "Auto-calibration of cylindrical multi-projector systems," in *Proc. IEEE Virtual Reality Conf.*, 2010, pp. 155–162.1

[28] B. Shneiderman, "Response time and display rate in human performance with computers," *Comput. Surv.*, vol. 16, no. 3, pp. 265–285, 1984.

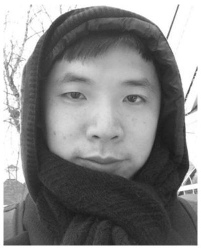
[29] M. Tchiboukdjian, V. Danjean, and B. Raffin, "Binary mesh partitioning for cache-efficient visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 5, pp. 815–828, Sep./Oct. 2010.

[30] C. Wang, J. Gao, and H. Shen, "Parallel multiresolution volume rendering of large data sets with error-guided load balancing," in *Proc. Eurographics Parallel Graph. Visualization Symp.*, 2004, pp. 23–30.

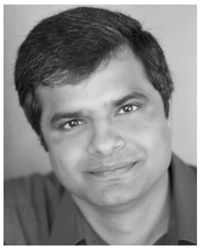
[31] S.-E. Yoon, B. Salomon, R. Gayle, and D. Manocha, "Quick-VDR: Out-of-core view-dependent rendering of gigantic models," *IEEE Trans. Vis. Comput. Graph.*, vol. 11, no. 4, pp. 369–382, Jul./Aug. 2005.

Duy-Quoc Lai received the MS degree from the School of Electrical Engineering at the University of California, Irvine. He is currently working toward the PhD degree in the School of Information and Computer Science, University of California, Irvine, CA. His research interests broadly include distributed system, parallel processing, multitiled display, and big data.

Behzad Sajadi received the PhD degree in computer science from the University of California, Irvine, CA, in 2012. He received the bachelor's degree from Sharif University of Technology in 2006. His main research areas were computer graphics and visualization with particular interest in multiprojector displays and computational projectors and cameras. He has several publications on geometric and photometric registration of multiprojector displays with the focus of practical designs for commodity large area displays. He has received the Best Paper award in Virtual Reality (VR) 2010 and the Second Best Paper award in Visualization 2009. He has also worked on novel camera and projector designs that provide new degrees of freedom to the users. Along this line, he has published a few papers two of which have appeared in Siggraph 2011 and Siggraph 2012.



Shan Jiang received the bachelor's of science degree from Cyprus College in 2007, and the PhD degree in computer science from the University of California, Irvine, in 2013. He is a software engineer in Altair Engineering Inc. His main responsibility is researching and developing modern graphics technology for the HyperView(R) product. Interactive visualization, massive model rendering, and postprocessing on CAD and CAE models are the primary fields he works on.



Gopi Meenakshisundaram (M. Gopi) received the BE degree from Thiagarajar College of Engineering, Madurai, the MS degree from the Indian Institute of Science, Bangalore, and the PhD degree from the University of North Carolina at Chapel Hill. He is a professor of computer science in the Department of Computer Science, University of California, Irvine. His research interests include geometry and topology in computer graphics, massive geometry data management for interactive rendering, and biomedical sensors, data processing, and visualization. His work on representation of manifolds using single triangle strip, hierarchyless simplification of triangulated manifolds, use of redundant representation for big data for interactive rendering, and biomedical image processing have received critical acclaim including best paper awards in two Eurographics conferences and in ICVGIP. He served as the program co-chair and papers co-chair of ACM Interactive 3D Graphics conference in 2012 and 2013, respectively, area chair for ICVGIP in 2010 and 2012, a program co-chair for International Symposium on Visual Computing 2006, and an associate editor of the *Journal of Graphical Models.* He is a gold medalist for academic excellence at Thiagarajar College of Engineering. He received the Excellence in Teaching Award at UCI and a Link Foundation Fellow. He is a member of the IEEE and ACM.

Aditi Majumder received the PhD degree from the Department of Computer Science, University of North Carolina at Chapel Hill in 2003. She is a professor of computer science in the University of California, Irvine. Her research resides at the junction of computer graphics, vision, visualization, and human-computer interaction. Her research focuses on novel displays and camera exploring new degrees of freedom and quality while keeping them truly commodity, easily accessible to the common man. She has received three Best Paper Awards in 2009-2010 in premier venues of IEEE Visualization, IEEE VR, and IEEE PROCAMS. She is the coauthor of the book *Practical Multi-Projector Display Design*. She was the program cochair of ACM Virtual Reality Software and Technology 2014, a general cochair of IEEE Virtual Reality 2012, a program co-chair of IEEE Virtual Reality 2011, a conference cochair for ACM Virtual Reality Software and Technology 2007, a general cochair of the Projector-Camera Workshop (PROCAMS) 2005, and the program chair of PROCAMS 2009. She has played a key role in developing the first curved screen multiprojector display that was marketed by NEC/Alienware. She has advised Disney Imagineering for advances in their projection based theme park rides. She has received the Faculty Research Incentive Award in 2009 and Faculty Research Midcareer Award in 2011 in the School of Information and Computer Science in UCI. She received the US National Science Foundation (NSF) CAREER Award in 2009 for ubiquitous displays via a distributed framework.