UNIVERSITY OF CALIFORNIA,
IRVINE

A Distributed Architecture for Interactive Scene Navigation, Modification, and Collaboration on Multi-Display Walls

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Duy-Quoc Lai

Dissertation Committee:
Professor Aditi Majumder, Chair
Professor Gopi Meenakshisundaram
Professor Ramesh Jain
Professor Nalini Venkatasubramanian

2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank Dr. David Kasik from Boeing for helping obtain the 20 GB Boeing model. It has been a wonderful test model to experiment with our architecture. I would like to thank Prof. Sandy Irani of the University of California, Irvine for helping me with the theoretical analysis on proximity of optimality achieved by my interleaved data partitioning method.

CURRICULUM VITAE

Duy-Quoc Lai

REFEREED JOURNAL PUBLICATIONS

A Distributed Memory Hierarchy and Data Management for Interactive Scene Navigation and Modification on Tiled Display Walls — **2015**
IEEE Transactions on Visualization and Computer Graphics

REFEREED CONFERENCE PUBLICATIONS

Interactive Display Conglomeration on the Wall — **March 2015**
IEEE VR2015

A Collaborative SAR Workspace for Modeling of Massive 3D Data — **March 2016**
IEEE VR2016 (Submitted)

TECHNICAL DEMOS

A Multi-Projector Display System of Arbitrary shape, Size and Resolution — **March 2015**
IEEE VR2015

A Multi-Projector Display System of Arbitrary Shape, Size, and Resolution (Enhanced) — **August 2015**
ACM Siggraph 2016

ABSTRACT OF THE DISSERTATION

A Distributed Architecture for Interactive Scene Navigation, Modification, and Collaboration on Multi-Display Walls

By

Duy-Quoc Lai

Doctor of Philosophy in Computer Science

University of California, Irvine, 2015

Professor Aditi Majumder, Chair

This dissertation addresses a growing challenge of visualizing and modifying massive 3D geometric models in a collaborative workspace by presenting a new scalable data partitioning algorithm in conjunction with a robust system architecture. The goal is to motivate the idea that utilizing a distributed architecture may solve many performance related challenges in visualization of large 3D data. Drawing data from modeling, simulation, interaction and data fusion to deliver a starting point for scientific discovery, we present a collaborative visual analytics framework providing the abilities to render, display and interact with data at a massive scale on high resolution collaborative display environments. This framework allows users to connect to data when it is needed, where it is needed, and in a format suitable for productivity while providing a means to interactively define a workspace that suits one's need. The presented framework uses a distributed architecture to display content on tiled display walls of arbitrary shape, size, and resolution. These techniques manage the data storage, the communication, and the interaction between many processing nodes that make up the display wall. This hides the complexity from the user while offering an intuitive mean to interact with the system. Multi-modal methods are presented that enables the user to interact with the system in a natural way from hand gesture to laser pointer. The

combination of this scalable display method with the natural interaction modality provides a robust foundation to facilitate a multitude of visualization and interaction applications.

The final output from the system is an image on a large display made up of either projection or lcd based displays. Such a system will have many different components working together in parallel to produce an output. By incorporating computer graphics theory with classical parallel processing techniques, performance limitations typically associated with the display of large or numerous items on multiple display devices and multiple input sources are overcome.

Chapter 1

Introduction

The discovery process is often based on collaborative visual analytics. Data is turned into visual information, which when properly presented and analyzed is turned into knowledge. Two essential factors help to accelerate this knowledge creation process: interactive visualization, which provides control over how data is processed and presented, and collaboration between different domain specialists who can build off of each others abilities to interpret and analyze the data.

Interactive visualization of data can be a significant challenge given the large amount of data that can be generated by modern data-driven science (e.g. geographic data, 3D model, and simulation). Facilitating collaboration also poses many difficulties due to inherent challenges in humain and computer interaction. Data sharing may not always be feasible because, the data is either too large, or the capabilities to process such vast amount of data are not available to collaborators. In these instances, distributing the derivative products, i.e. visuals extracted from the data, can be a more suitable and readily available alternative [39].

Large-high resolution visual real estate is needed to express the underlying data complexity without compromising context, and a large physical form factor is needed to support

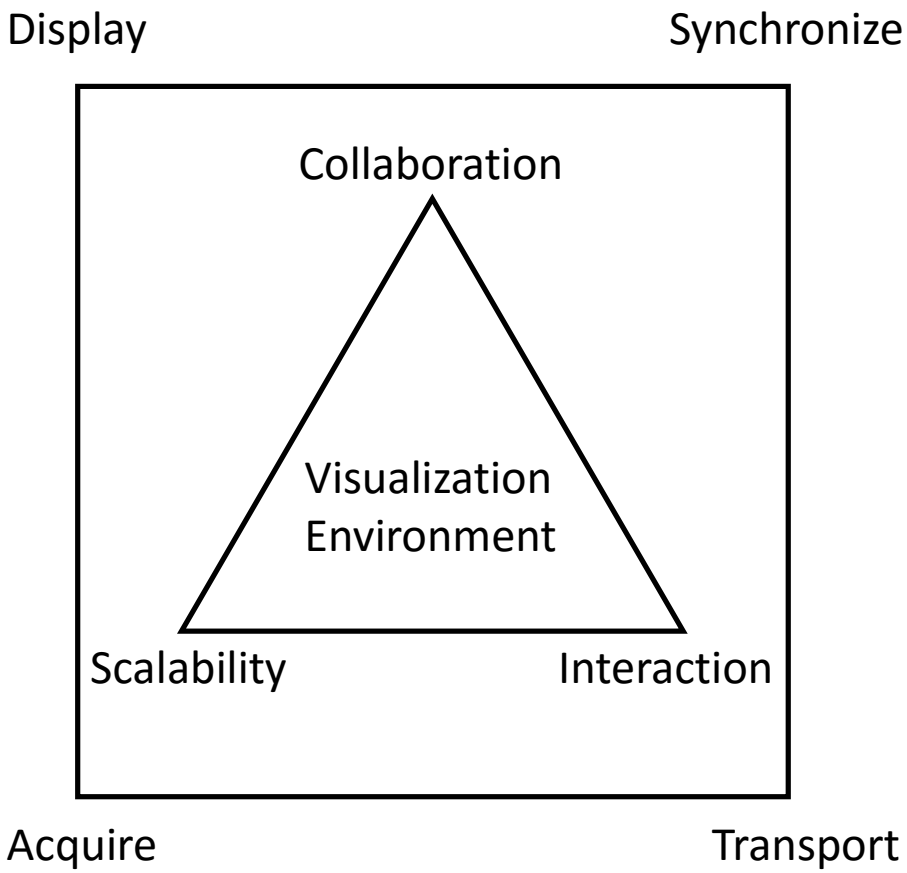


Figure 1.1: The three primary objectives of the presented framework are to improve collaboration, interaction, and scalability by using new methods to acquire, transport, synchronize, and display data at a massive scale.

collaborative teams. These requirements are currently achieved by utilizing room-size, high-resolution collaborative environments built from multi-tile LCDs and projectors in a variety of configurations to form display walls for 2D and 3D virtual reality environments.

This dissertation addresses a growing challenge of visualizing and modifying massive 3D geometric models in a collaborative workspace by presenting a new scalable data partitioning algorithm in conjunction with a robust system architecture. The main challenge is enabling simultaneous navigation and editing of large 3D geometric model while maintaining an interactive framerate. A massive 3D model is impossible to interact with using a

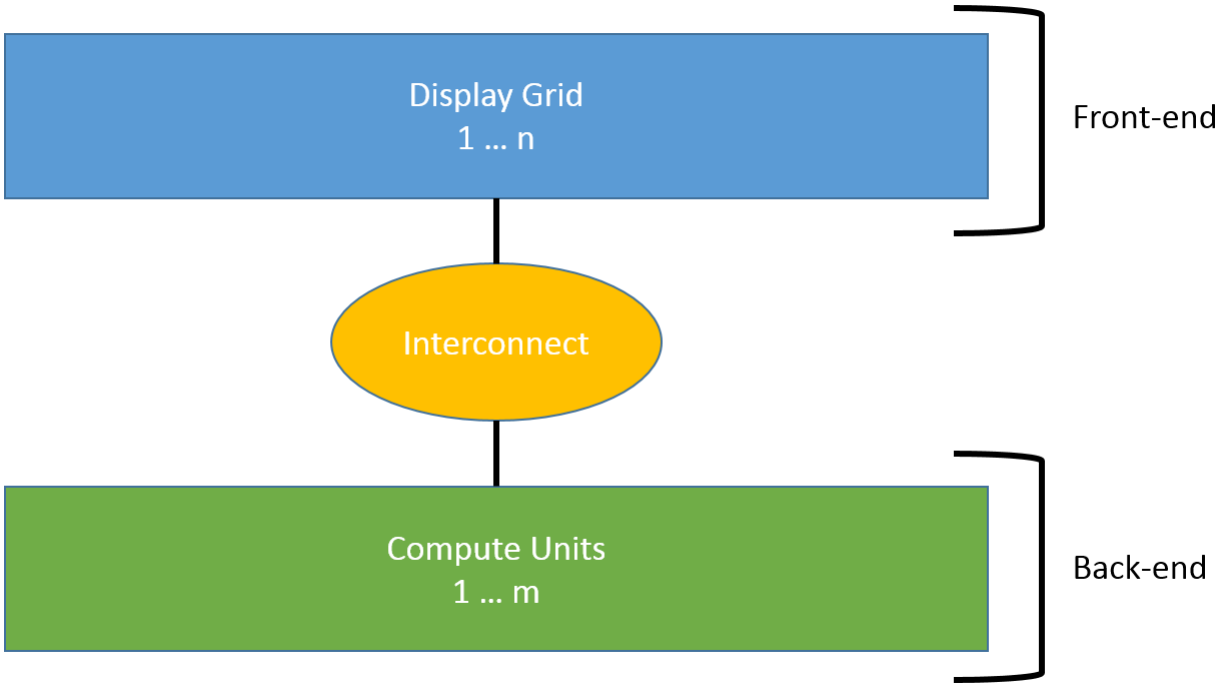


Figure 1.2: The front end consists of the display grid, which is made up of n displays. The back end consists of m compute units. Note that the number of displays and compute units do not have to be equal.

single computer. Therefore, a backbone of a conglomeration of computers is essential to do anything collaborative or interactive with such models. Furthermore, today's 3D models are getting exponentially large and complex with no collaborative tools to match its scale and complexity so that many users can work easily with such models. Drawing data from modeling, simulation, interaction and data fusion to deliver a starting point for creative discovery, I present a collaborative visual analytics framework providing the abilities to render, display and interact with data at a massive scale on high resolution collaborative display environments. This framework aims to improve collaboration, interaction, and scalability using a series of techniques to acquire, transport, synchronize, and display data at a massive scale on large-high resolution collaborative visualization environments, as outlined in Figure 1.1. This framework allows users to connect to data when it is needed, where it is needed, and in a format suitable for productivity while providing a means to interactively define a display workspace that suits one's need.

This dissertation is divided into two components: front end and back end. Figure 1.2 gives an illustration of how the two components are connected. The front end handles the collaboration and interaction objectives by providing a large seamless display with natural interaction modalities to hide the complexity of the system. The back end handles scalability objective by using a distributed architecture and incorporating computer graphics theory with classical parallel processing techniques to tackle performance limitations typically associated with the display of large or numerous items on multiple display devices and multiple input sources are overcome. Since a display wall is a system, and like all systems, to achieve optimal performance, there has to be a balance between different components, thus understanding the various types of interactions and dynamics within a system is key to increasing computer scientists understanding of how to build and maintain more efficient and effective systems.

The work presented in this dissertation focusses on identifying each component that makes up a display wall system and on designing a high performance, high resolution, and interactive visualization system without specialized or customized hardware that is scalable, flexible, reliable, usable, and affordable.

We begin with the introduction of the architecture. In chapter 2, we examine why a distributed architecture is ideal for creating a highly parallelizable and scalable display system. We look at how current display wall systems are implemented and show our system addresses certain challenges that other systems have not.

In chapter 3, we define a new memory hierarchy that is associated with the distributed architecture and how to leverage it to increase data acquisition performance when working with massive 3D geometric models. We also present a data layout and data management system to evaluate the performance of the architecture.

In chapter 4, we analyze an existing system to determine the effectiveness of the proposed memory hierarchy, data layout, and data management system.

In Chapter 5, we describe how the system capabilities can be used to promote collaboration. We look at how collaboration are typically done in the workplace today and how our system with its capabilities can enhance the collaboration experience and reduce the overhead that is commonly associated with collaborating and sharing resources.

Chapter 2

Distributed Architecture for Highly parallelizable and scalable Display System

2.1 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 [50].

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.

2.2 Sorting Classification of Parallel Rendering

There are three broad classes of parallel rendering methods, based on where the sort from object-space to screen space occurs. That is to say where the data distribution occurs during the process of transforming 3D geometric data to 2D pixel data. These classes encompass most feed-forward parallel software and hardware rendering architectures that have been described to date. Figure 2.1 shows a basic overview of the standard, feed-forward rendering pipeline, adapted for parallel rendering. The pipeline consists of two main parts: geometry processing, and rasterization. Geometry processing usually is paralelized by assigning each processor a subset of the primitives (objects) in the scene. During the geomtery processing stage, 3D geometric data is transformed based on the view point to meta-2D space (the depth

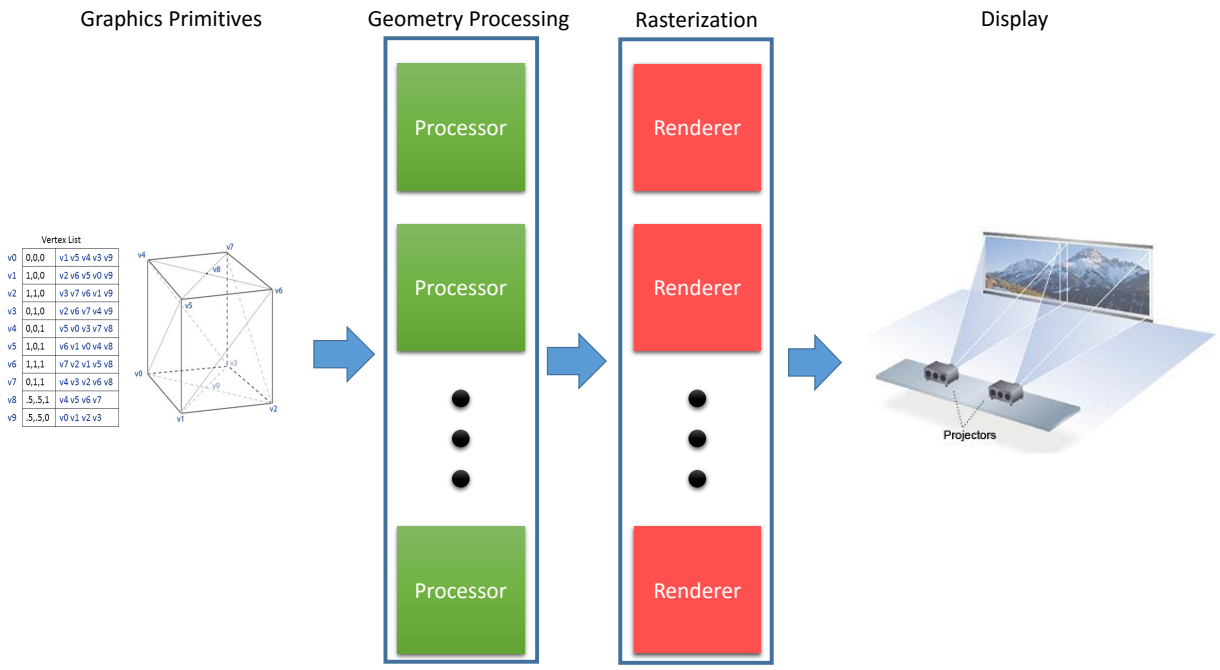


Figure 2.1: The graphics pipeline in a parallel rendering system. The processors perform geometry processing. The renderers perform rasterization.

is perserved). Rasterization usually is parallelized by assigning each renderer a portion of the pixel calculations. During the rasterization stage, the meta-2D data is transformed into 2D data, pixel, so that it can be displayed. The sorting can occur before the geometry processing stage, in between the geometry processing and the rasterization stage, or after the rasterization stage. Molnar et al. [32] have defined the three occurrences: sort-first, sort-middle, and sort-last .

2.2.1 Sort-first

In a sort-first method, primitives are distributed early in the rendering pipeline, before geometry processing, to processors (a.k.a. renderers) that require the data to perform the reamining rendering calculations (Figure sort first). This is done by dividing the display screen into disjoint regions and designating processors for all rendering calculations that

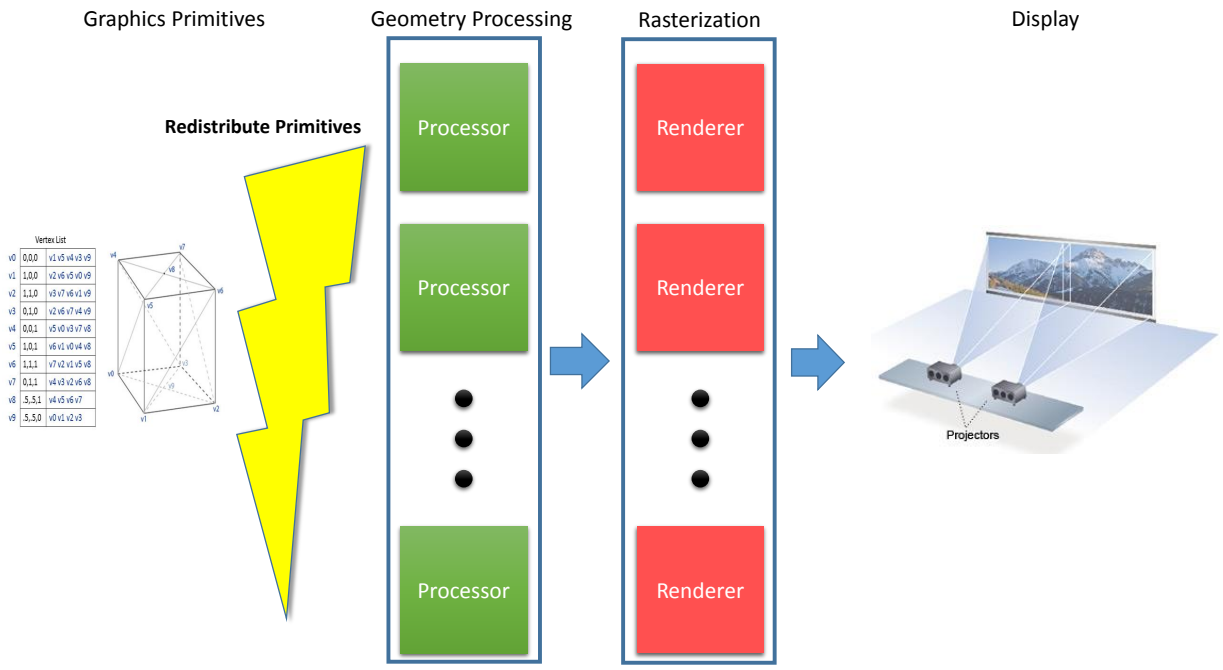


Figure 2.2: Sort-first method. Redistributes raw primitives prior to the geometry processing.

affect their respective screen regions. As a pre-process, primitives are stored either remotely or locally on renderers in some arbitrary fashion. At the start of the rendering process, each renderer determines the region of the scene they need to process and fetch all primitives that fall into that region. This is generally done by computing the screen-space bounding box of the primitive. Based upon how the primitives are initially laid out, the redistribution of primitives to the renderers may drastically affect overall rendering performance.

This redistribution of primitives at the start of the rendering process is the main feature that distinguishes a sort-first method. The main disadvantage of the sort-first method is that it is scene dependent. Given a multi-renderer system, a renderer can be tasked to render more data than other renderers, thus affecting the number of frames per second that it can output. Although sort-first may appear impractical at first due to the rendering load balancing issue, it uses much less communication bandwidth than the other approaches, especially if spatial locality are utilized or if frame-to-frame coherence can be exploited. It is also important to

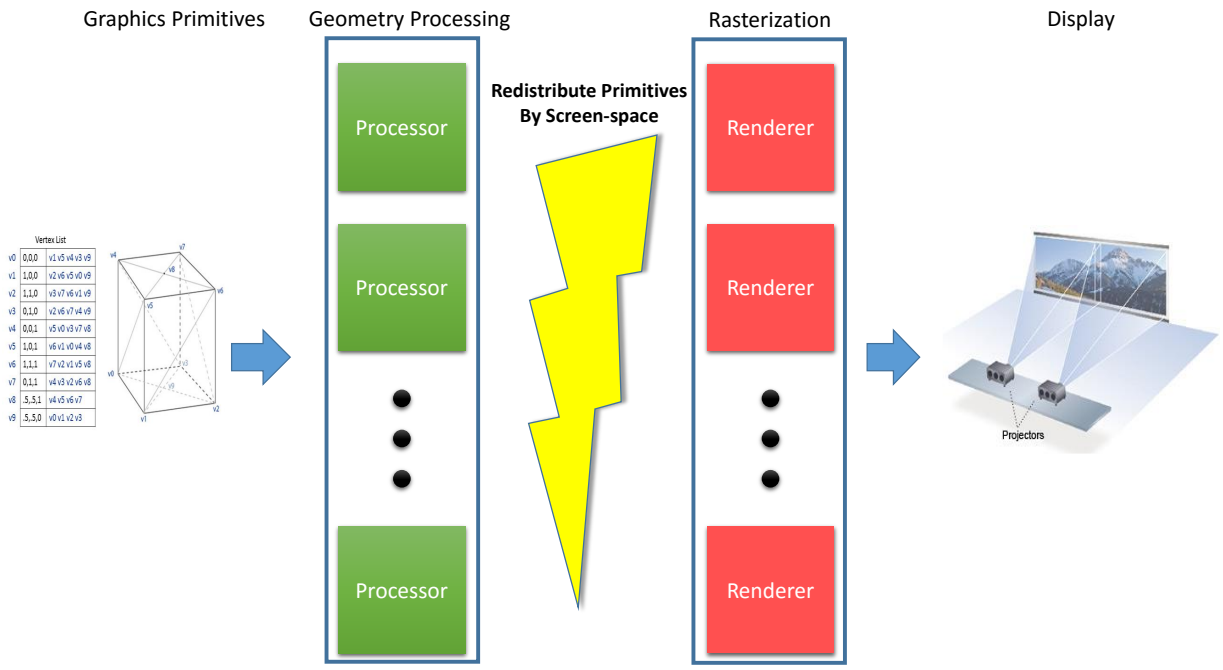


Figure 2.3: Sort-middle method. Redistributes the primitives based on the screen-space. This occurs between the geometry processing and rasterization stage.

note that as the number of renderers increases the sort-first method scales since the same amount of data is rendered by more renderers, thus reducing the average workload.

2.2.2 Sort-middle

In the sort-middle method, primitives are redistributed in the middle of the rendering pipeline, between the geometry processing stage and the rasterization stage. Geometry processing and rasterization can be performed on separate processors or on the same processor but on separate components of the processor, hence the separation can occur between the two stages.

In a sort-middle system, geometry processors are allocated to a subset of the primitives to be displayed and rasterizers are assigned a portion of the display screen. During the rendering

process for each frame, the geometry processors transform, apply lightning... to all primitives allocated to them and organize them with respect to the screen space. The geometry processors then transmit all of the screen-space primitives to the appropriate rasterizer(s).

Since the two processors (geometry and rasterizer) may be separate sets of processor transition between the two stages of the pipeline is well integrated into the hardware process. A possible way to implement the sort-middle method is to have two graphics cards and dedicate one graphics card to processing the geometry and transferring the screen-space primitive to the second graphics card for rasterrs, or they may time-share the same physical processor.

Sort-middle is general and simple, and has been explored in both software [17, 60] and hardware [14, 1, 21] parallel rendering systems.

2.2.3 Sort-last

In the sort-last method, the sorting is done at the end of the rendering pipeline, when all the primitives have been rasterized into pixels. In sort-last the processors are referred to as renderers. Each processor is assigned arbitrary subsets of the primitives. Each processor computes pixel values for its subset, regardless of its actual location in the display screen. Renderers then transmit these pixels over a communication network to compositing processors, which processes the visibility of pixels from each renderer.

The visibility check stage is the last stage in the rendering pipeline. Renderers operate independently up until the visibility check stage. After the visibility check stage, the pixels must be transferred to designated display processors over the communication network. For real-time application rendering high-resolution images, this can result in significantly high data rates.

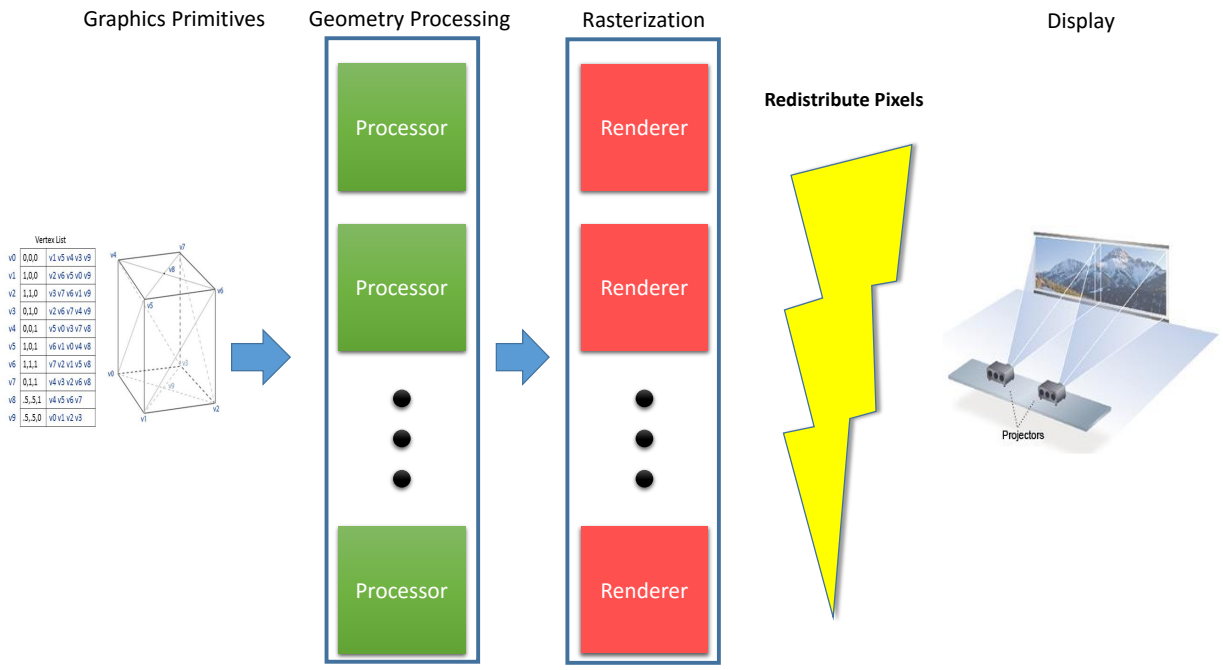


Figure 2.4: Sort-last method. Redistributes the pixels to the display.

A sort-last system can be implemented in two ways. One method minimizes the amount of data transferred by only distributing pixels that are actually produced by rasterization. This is method is known as *sparse*. The second method transfers a full image from each renderer, where null pixels will be interpreted as a valid pixel with an alpha for selective composite. This method is known as *full-frame*. The second method is straightforward and can easily embedded in a linear network and scales linearly. Since the renderers implement the entire rendering pipeline up until the final pixel merging stage, they can operate independently until then. This leads to a better load distribution across all renderers. Since the rendering can be done on any renderer and not be tied to the screen space, unlike sort-first. There is a large body of work in the area of sort-last. Molnar et al. presents a system that handles primitive-per-processors [34] and Shaw et al. presents a system that handles multiple-primitives-per-processor [45].

2.3 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 manuscript. There exists a plethora of work in distributed rendering paradigms like SAGE, Equalizer, Chromium and Gigawalk [43, 16, 26, 2], 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 [26, 2, 42] 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 [26, 2] or can be pre-organized using any particular data structure, such as a graph [42], a space filling curve [59, 6], or an octree [4]. When there is a request from the application, this data structure can then be traversed to locate the requested data [47, 25]. 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 [16, 43] follows a sort-last architecture, although [16] 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, [16, 43] 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 [16] leaves the data distribution approach open to the application.

Recent work like the ones proposed by Erol et al. [18], and Moloney et al. [35] 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 [33] 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.[36], Chapman et al. [9], and Fitzpatrick et al. [20]. 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. [50] have proposed solutions for interactive edits while maintaining interactive rendering speeds for Solid State 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 [23], Quick-VDR [61], and other cache-oblivious data layouts [19, 57, 30, 4]. 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.

Chapter 3

Distributed Memory Hierarchy

We propose a distributed infrastructure and memory architecture for navigation and modeling, using a tiled cluster of SCD nodes, as shown in Figure 3.1. 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. [3], and Sajadi et al. [52, 51, 53]. Figure 3.2 shows such an adjacency graph for a 3×3 array of 9 projectors, the prototype system we use in this manuscript.

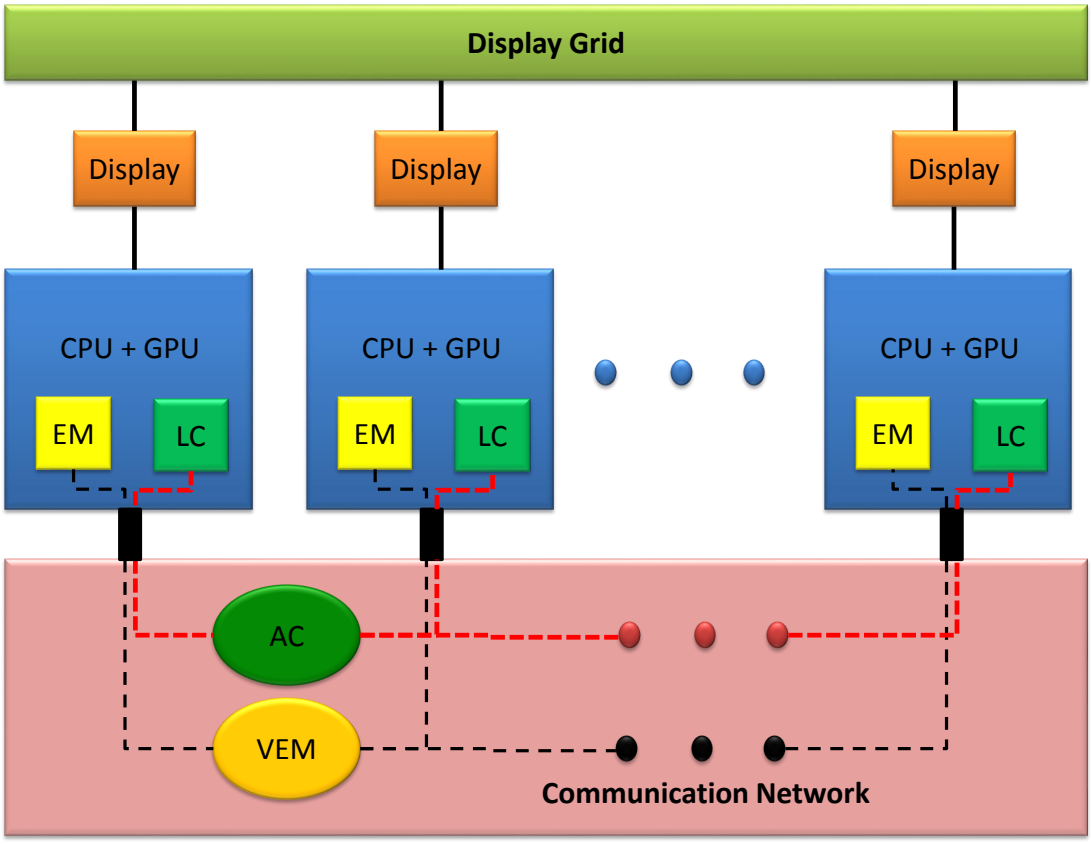


Figure 3.1: *Proposed Distributed Memory Hierarchy. The distributed architecture utilizes a cluster of PCs to define a new memory hierarchy - from the fastest to the slowest – the Local Cache (LC), Adjacency Cache (AC), Local External Memory (EM), and Virtual External Memory (VEM). When the data are not found in a faster memory, it is searched in and fetched from the slower levels of memory hierarchy.*

3.1 Memory Hierarchy

We denote the secondary storage of each SCD node as EM (external memory) and the local main memory as LC (local cache). For each SCD node, the collection of LC of adjacent SCD nodes, as given by the adjacency graph is termed AC (adjacency cache). The dashed red line in Figure 3.1 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 VEM (virtual external memory) is the combination of all the other SCD nodes' EM, not

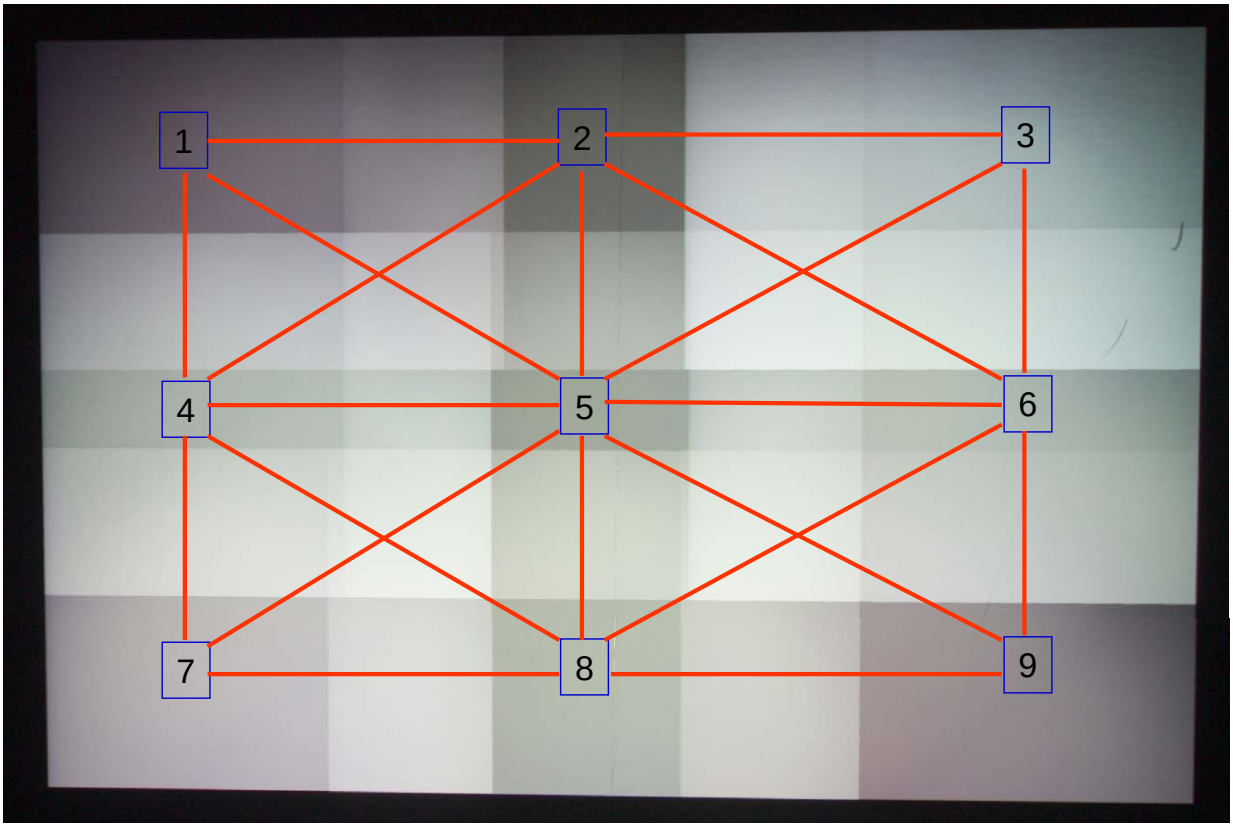


Figure 3.2: The grid connection formed by the multi-display via spatial overlaps.

just those of the adjacent SCD nodes. The dashed black line in Figure 3.1 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.

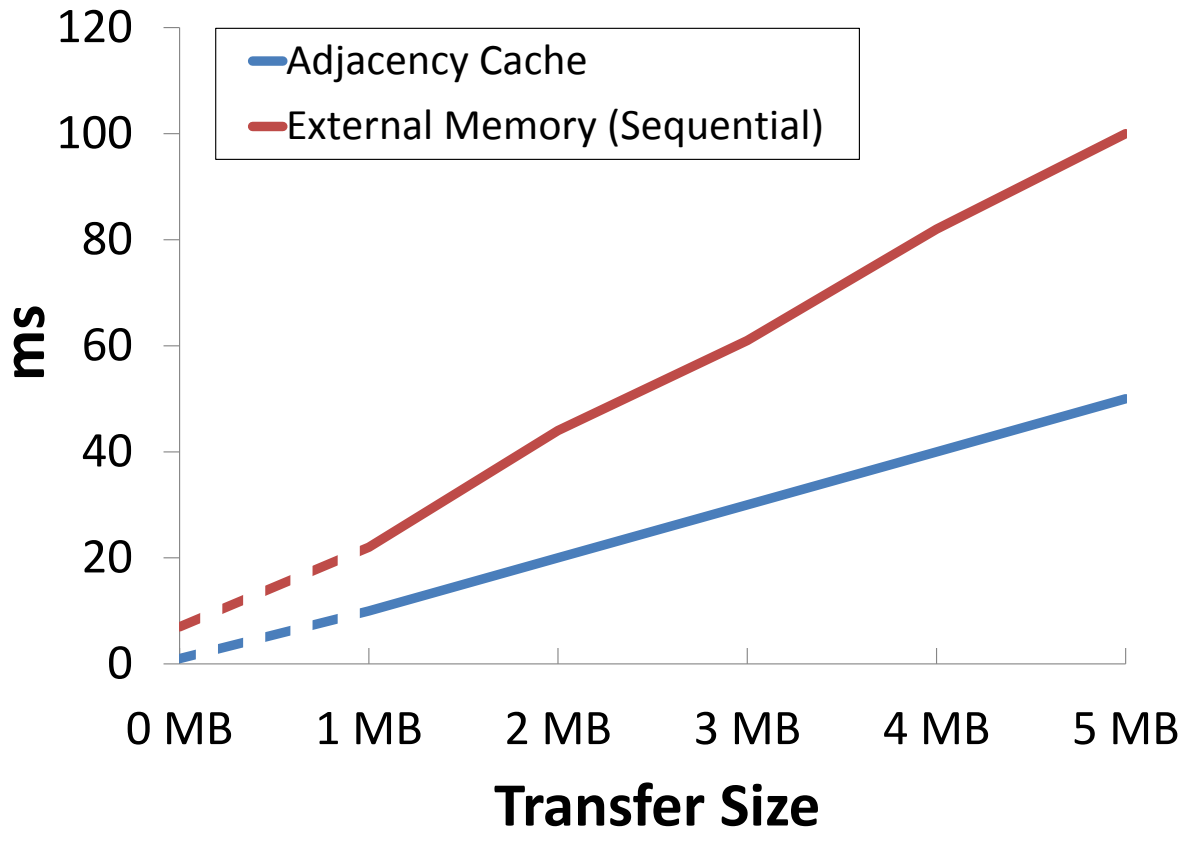


Figure 3.3: Comparison between the transfer time for the AC and the EM, for different transfer size.

Figure 3.3 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 Figure 3.3 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 Figure 3.3, 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 Figure 3.3 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 vs. 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

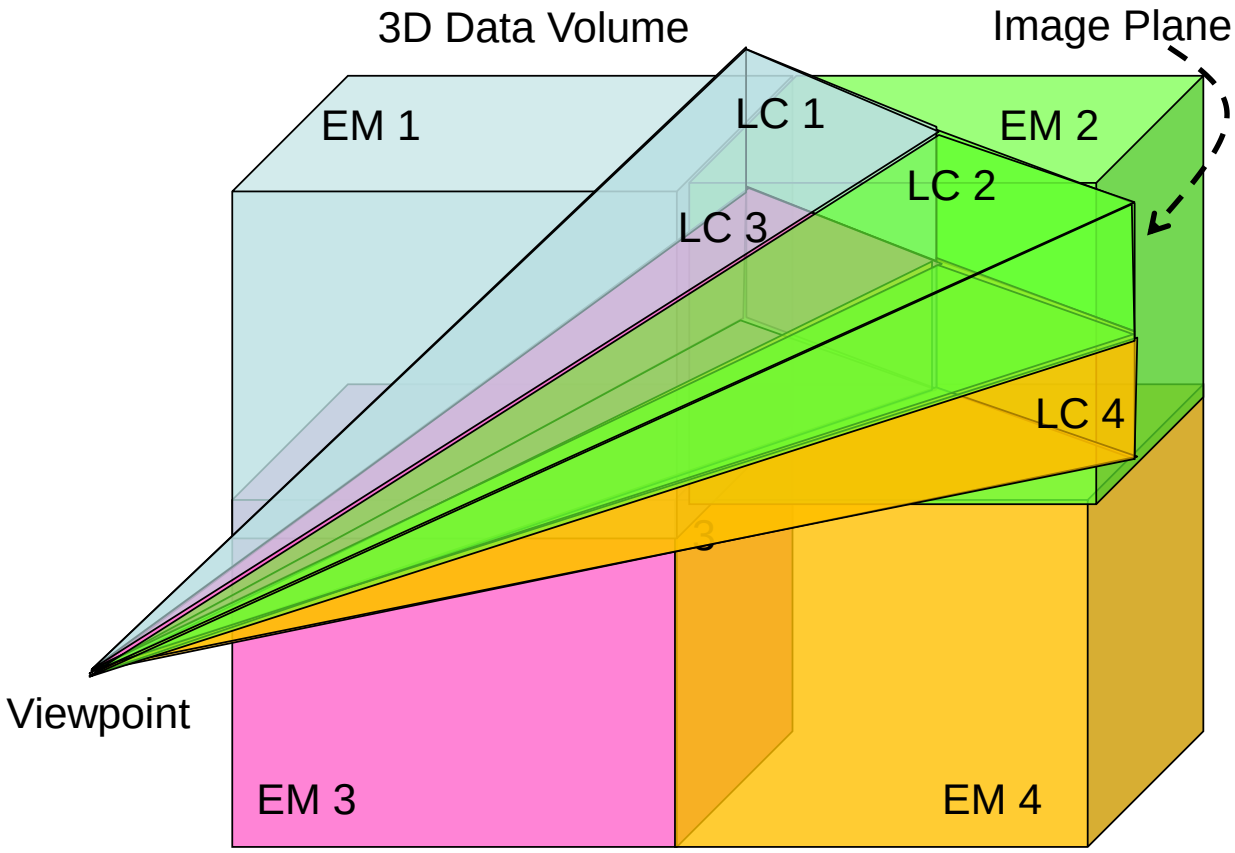


Figure 3.4: An example of data partitioning and retrieval for a system composed of 4 SCD nodes: The 3D data volume is partitioned and stored on the EM of the four machines (colored differently in the figure). The content that an SCD node displays is dependent on the view point and view frustum, rather 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.

data to different VEMs, without duplication. Hence, we propose schemes for *partitioning* the complete data set in the object space and store the partitions in participating EMs 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 (Figure 3.4), 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.

3.4 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 (Figure 3.5), which is described in detail in Section 4. 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.

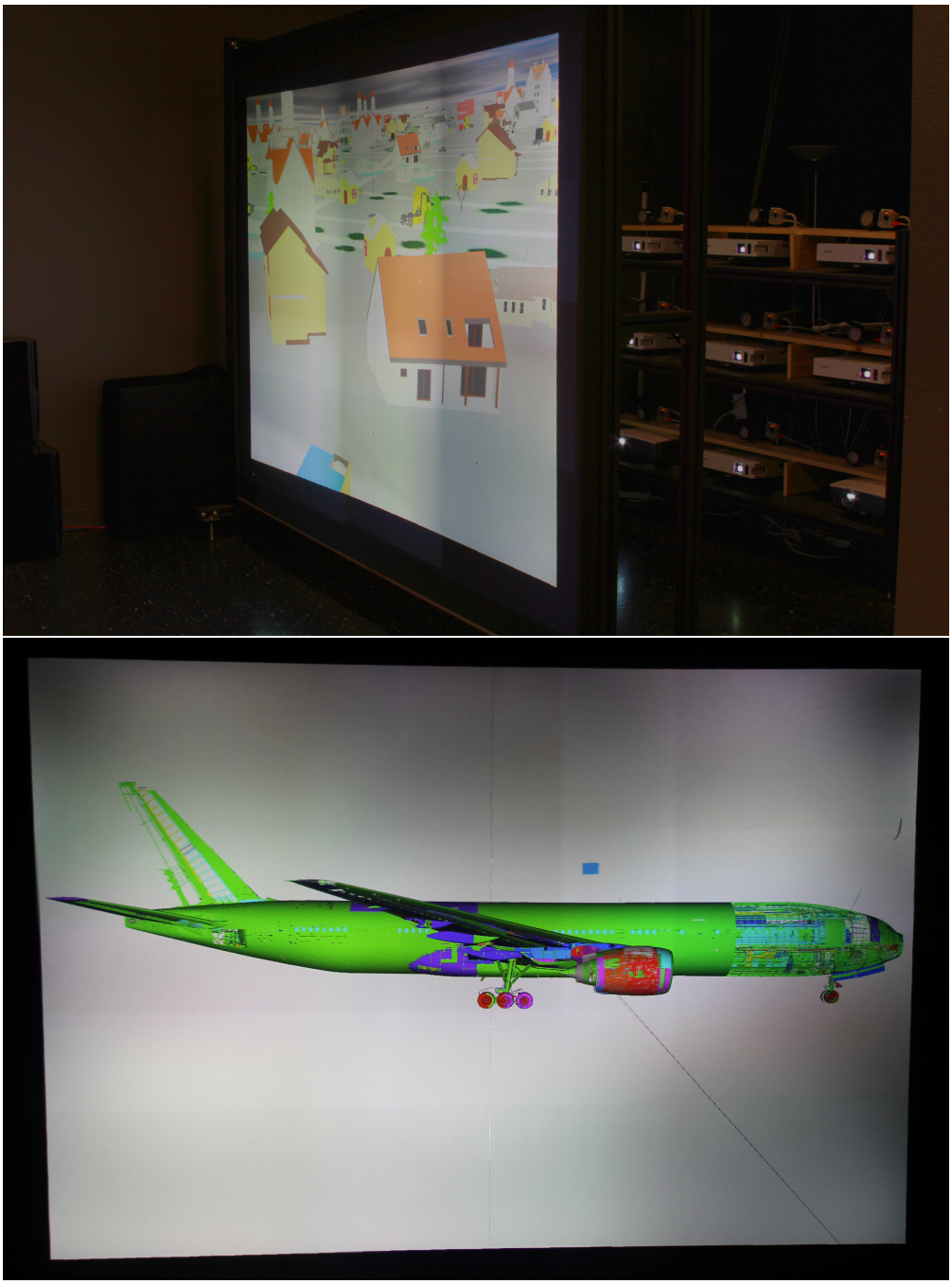


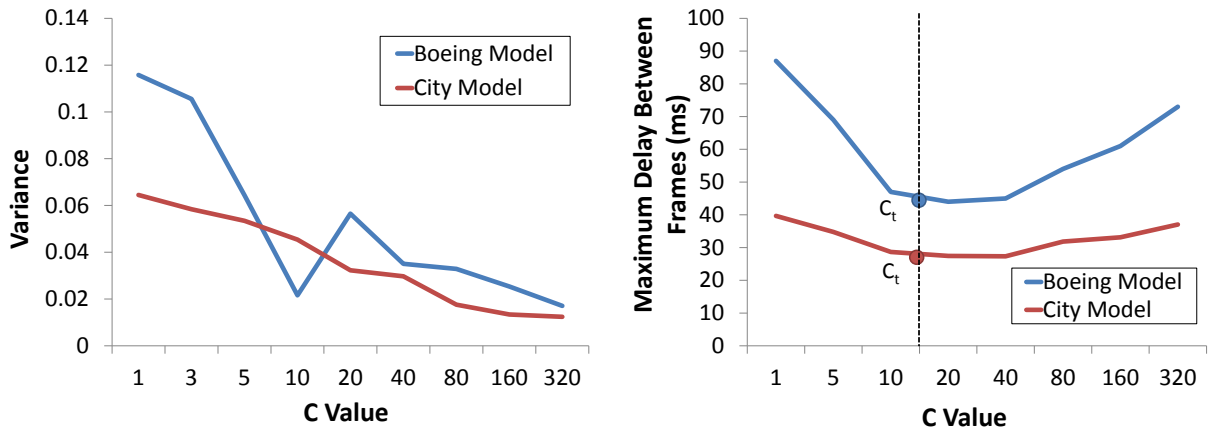
Figure 3.5: Top: This shows a display of the City model on our 3×3 grid of storage-compute-display (SCD) nodes, projected from a 3×3 multi-projector display. Bottom: A 20 GB Boeing 777 model, displayed on the 3 x 3 SCD grid.

3.4.1 Interleaved Data Partitioning

In this section we describe the modulo interleaving method, an interleaved data partitioning scheme for the 3D data, on a 2D grid, such as those commonly used in walk through 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 walk through 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 $0 \leq k \leq c-1$. Figure 3.4 shows an example for $m = n = 2$ and $c = 1$. Figure 3.9 shows data distribution for $m = n = c = 3$. Data interleaving based on space filling curves [59, 6], 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). Figure 3.10 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



(a) (b)

Figure 3.6: *(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 Figure 3.9. (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*.*

consider the Boeing model, a highly non-uniformly distributed data set. Figure 3.6a 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 Figure 3.6a. 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 3.4.2.

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

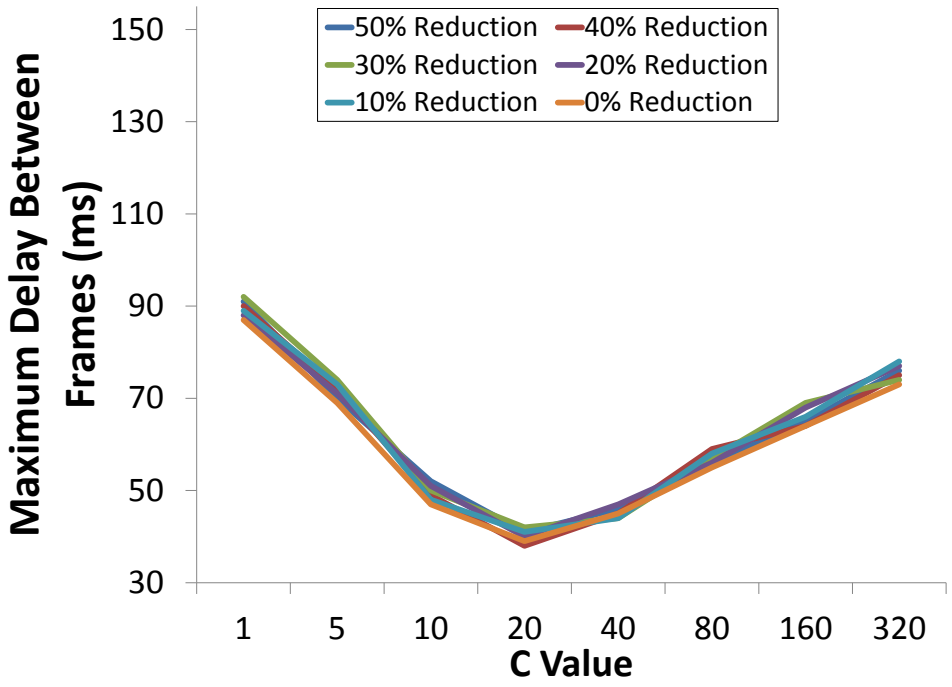


Figure 3.7: *Scalability of the Proposed Data Distribution Method. The performance of the system remains the same as the size of the data, the disk transfer time, and the network bandwidth are reduced by the same percentage. As the data size increases and the hardware improves, our data distribution scheme scales with the new platform by efficiently utilizing all the SCDs to avoid bottlenecks and to scale linearly.*

network bottleneck. Figure 3.6b 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 1000 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 (Figure 3.7). 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 Figure 3.6b, 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.

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. [12]. It has been shown by Graham [24], 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 Figure 3.9. 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

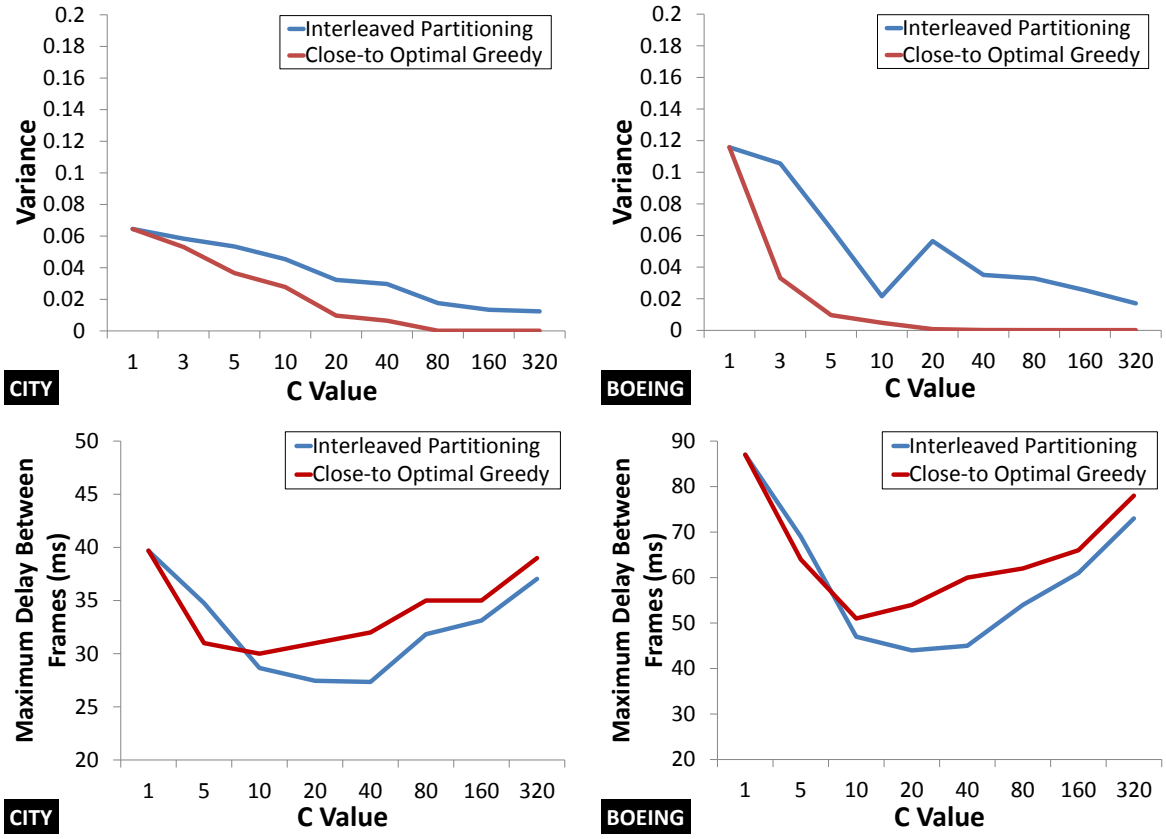


Figure 3.8: *This figure compares the variance (top) of the data distribution and the maximum delay between frames (bottom) of our interleaved data distribution (blue), with respect to a close-to-optimal greedy approach [24] (red) – for the City (left) and Boeing (right) model. Our interleaved distribution achieves a data load balancing close to this greedy approach. It also shows significantly lower maximum delay, benefiting from a spatially coherent layout.*

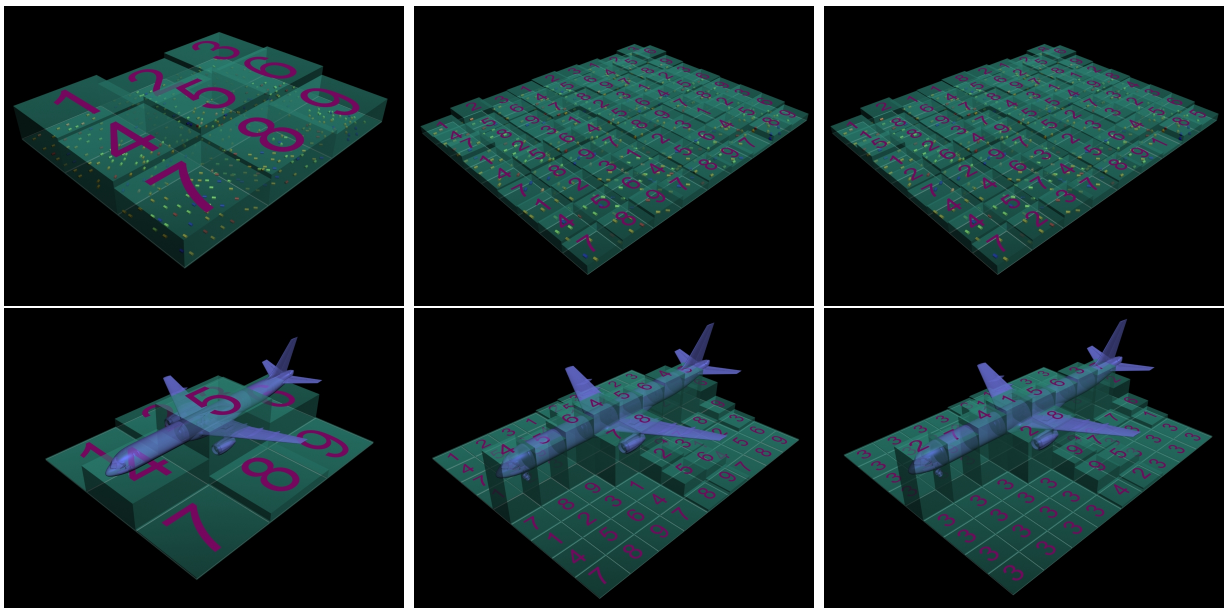
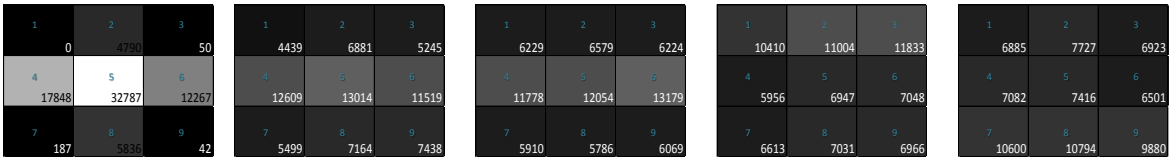


Figure 3.9: *This figure illustrates the interleaved data partitioning method on a* 3×3 *array of SCD nodes, for a 350 million triangles, 20 GB, Boeing 777 model (bottom), and a 4 GB, 110 million triangles, City model (top). The cells show the spatial partitioning. The height of the cell denotes the 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.*



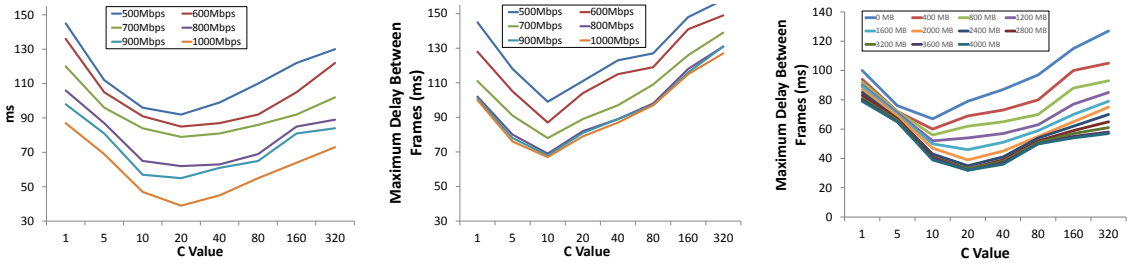
(a) $c = 1$ (b) $c = 5$ (c) $c = 20$ (d) $c = 40$ (e) $c = 160$

Figure 3.10: *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.*

of the view frustums. Figure 3.8 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. Figure 3.8 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 (Figure 3.9) 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.

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 (Figure 3.11a), (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 (Figure 3.11c). The results



(a) (b) (c)

Figure 3.11: *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 (Figures a and b), and different size for the AC (Figure 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.*

of these experiments are shown in Figure 3.11. The first two experiments ((a) and (b)) were performed with different network bandwidths, ranging between 500–1000 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.

- Overall, Figure 3.11a's maximum delay between frames is less than that in Figure 3.11b's. This is because Figure 3.11b 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 Figure 3.11a, 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 Figure 3.11b than in Figure 3.11a. 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 (Figure 3.11b), this delay is more pronounced than in the scenario where the AC is used (Figure 3.11a).

- Figure 3.11b 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 Figure 3.11a.

- Figure 3.11c shows that the performance of the system does improve with larger AC, since it will result in fewer cache misses.

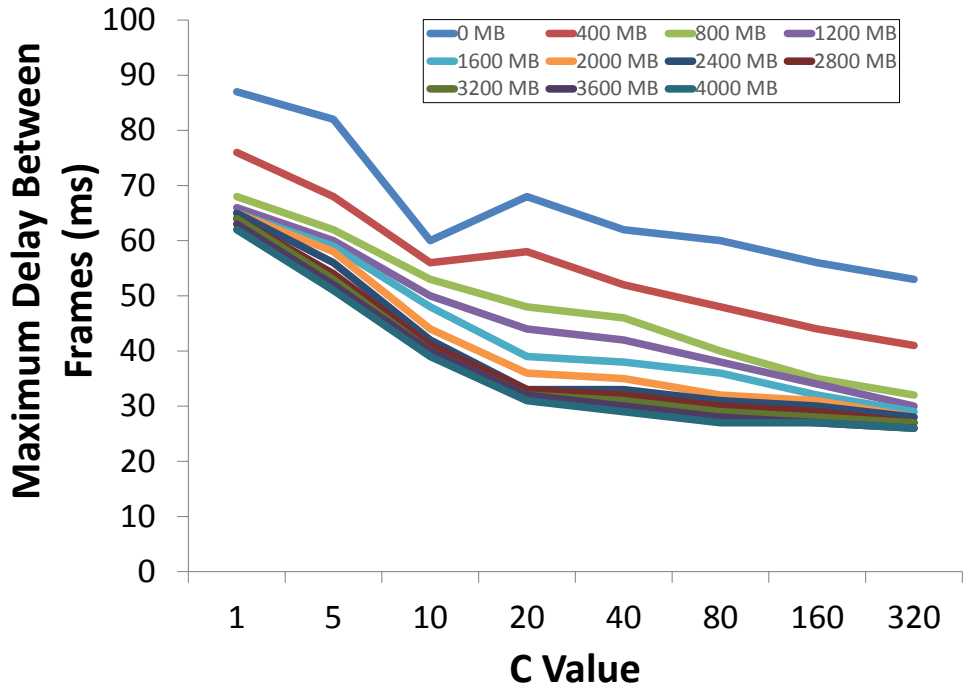


Figure 3.12: *This plot shows the maximum frame delay when an SSD is used, in place of a HDD, for different AC size. The network bandwidth, data set size, and disk transfer rate remain the same.*

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 Figure 3.6, 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 Figure 3.11b when compared to Figure 3.11a). 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 Figure 3.12 and this explains how SSDs performs differently than HDDs. The parameters for the experiment are the same as the ones used to generate Figure 3.11c. Figure 3.12 shows that the maximum delay monotonically decreases when SSDs were utilized, compared to the high delay shown in Figure 3.11c, 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 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 Figure 3.11c shows the behavior of HDDs for this scenario. Notice the difference between this and the curve in Figure 3.12. 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 Figure 3.6a, 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 Figure 3.12 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.

Figure 3.12 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 Figure 3.12, where the curves form a cluster, when the AC size is greater than 2000 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 Figure 3.8. 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.

3.4.2 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 Figure 3.8. 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 Figure 3.13. 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.

Figure 3.13, left, illustrates the initial bounding boxes. The data contained in these four bounding boxes can be mapped to a 2x2 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 Figure 13 right, will either contain more or fewer primitives than before, and when these data are mapped to 2x2 SCD nodes, we achieve better data balancing across nodes. Figure 14, illustrates the entire algorithm that has four iterations of the above processing shown in Figure 13, but on a 3x3 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 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 Figure 3.14. Each iteration of our load balancing procedure

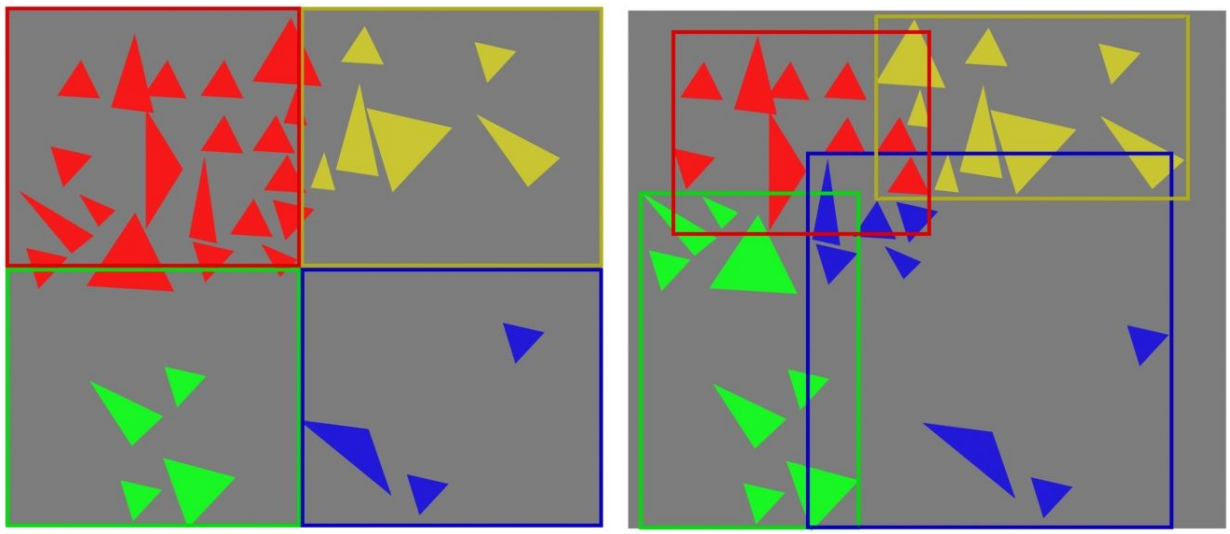


Figure 3.13: *In scene adaptive load balancing, cells are allowed to grow or shrink in a constrained manner (to retain most of the spatial coherence), while overlapping with their adjacent neighbors to achieve better data load balancing. The left shows four different colored cells (containing primitives 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.*

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

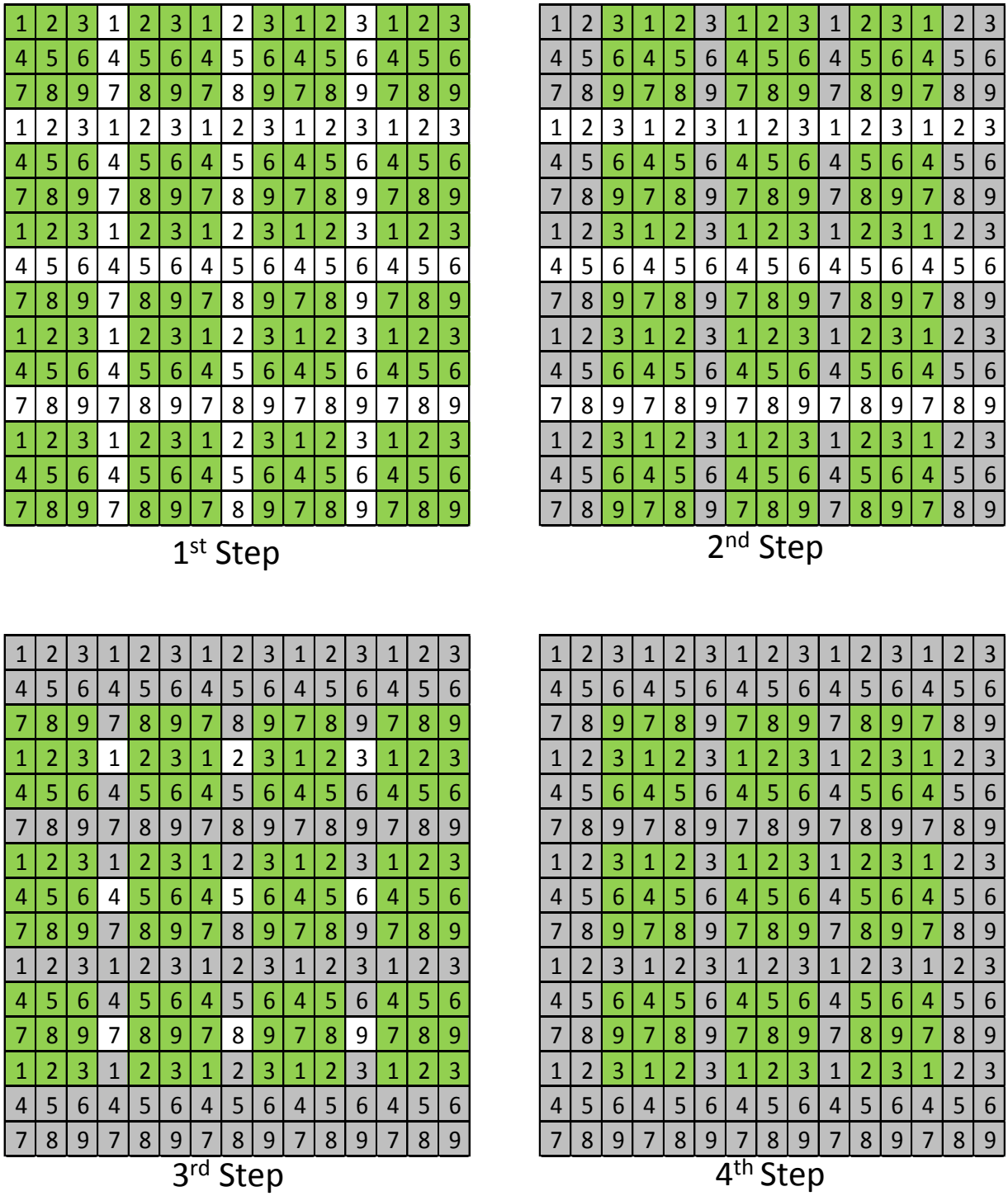


Figure 3.14: *A* 3×3 *array of machines, with interleaving of* $c = 5$*, resulting in* 15×15 *cells. This figure shows the different metacells processed, in parallel, for each step of the scene-adaptive balancing scheme. The cells not covered by a metacell in any of the previous steps is shown in white. Note that in four steps, all cells are processed.*

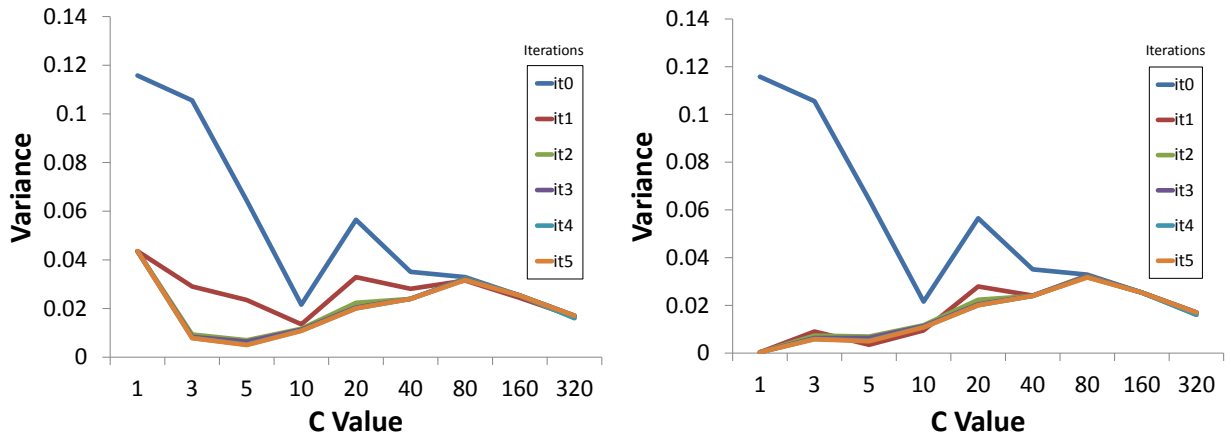


Figure 3.15: *The variance in the data load is reduced over multiple iterations of the scene-adaptive balancing scheme. This figure shows the change in the variance for different values of c, and for the bounding-box scale constraint of $f = 0.5$ (left), and $f = 1$ (right), for the Boeing Model. Note that as f increases, relaxing the spatial coherence constraint, the variance in the load is further reduced.*

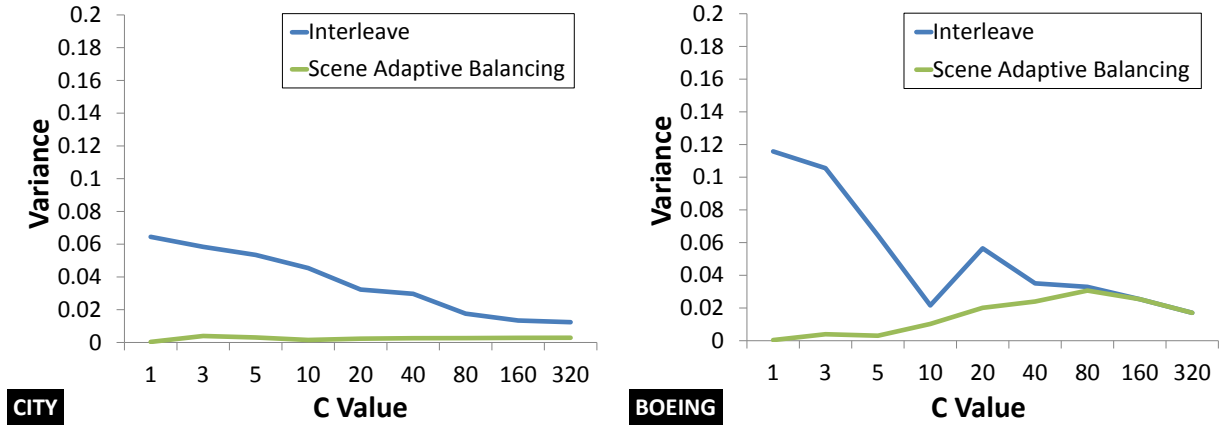


Figure 3.16: *The variance of the data load, resulting from the scene adaptive data balancing scheme (in bolded green lines), is much smaller than the interleaved approach (in bolded blue line), both for the City (left) and the Boeing (right) model. Note that, by design, the variance for the scene adaptive balancing scheme cannot be worse than the interleaved approach.*

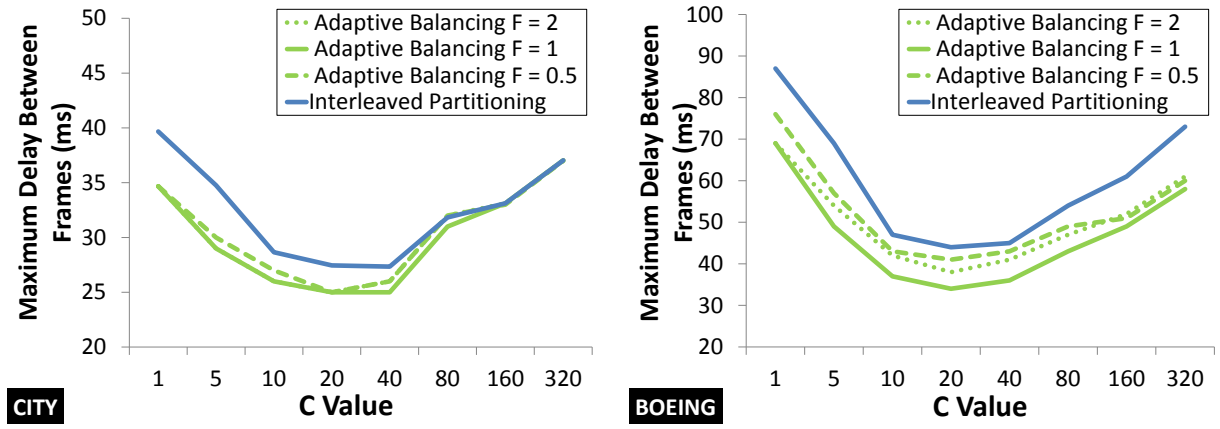


Figure 3.17: *This figure shows the effect of the scene adaptive data balancing scheme for different f ($f = 0.5, 1.0$, and 2.0), compared to the maximum delay between frames, for the City (left), and Boeing model (right).*

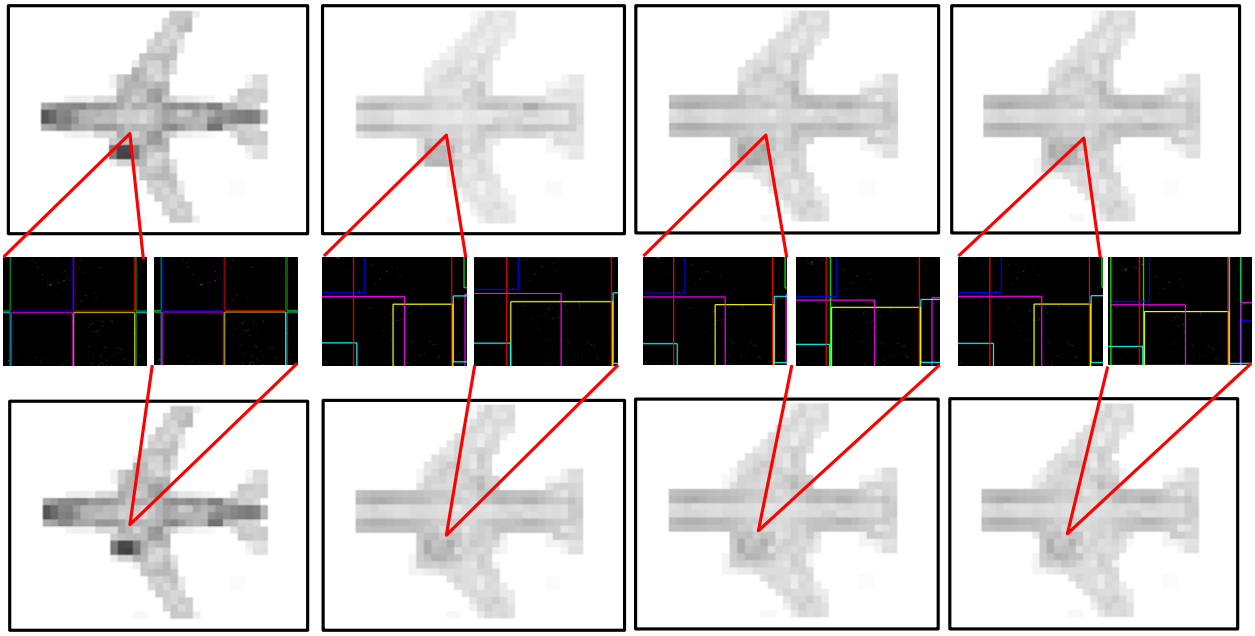


Figure 3.18: *This figure is a pictorial representation of the amount of data in each cell for $c = 5$, for a 3×3 system, where the amount of data is represented by a gray scale value. From left to right, we show the results after 0, 1, 5 and 23 iterations, respectively. Since the variation in the data 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.*

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 ← 4
BLK ← Blocks inside grid
D ← Maximum distance between cell
NC ← Number of cells in block
for all StepCount do
  for all BLK do
    Sort cell by data size
    T ← total data size of all cells
    AVGSIZE ← T/NC
    DELTA ← A non-negative number (∼ 5% of the AVGSIZE)
    while not All cells visited do
      C ←cell with the least amount of data
      OC ←all other cells with more data than C
      CLOAD ←data size of C below AVGSIZE - DELTA
      if CLOAD < 0 then
        for all OC do
          if OC's distance from C < D then
            OCLOAD ←data size of OC above AVGSIZE + DELTA
            if OCLOAD > 0 then
              MIN ← 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
```

Figures 3.15-3.17 presents the results for this scene adaptive data distribution scheme. Figure 3.15 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. Figure 3.16 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, Figure 3.17 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. 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.

Figure 3.18 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 Figure 3.17, 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.

3.5 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 Figure 3.4, 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

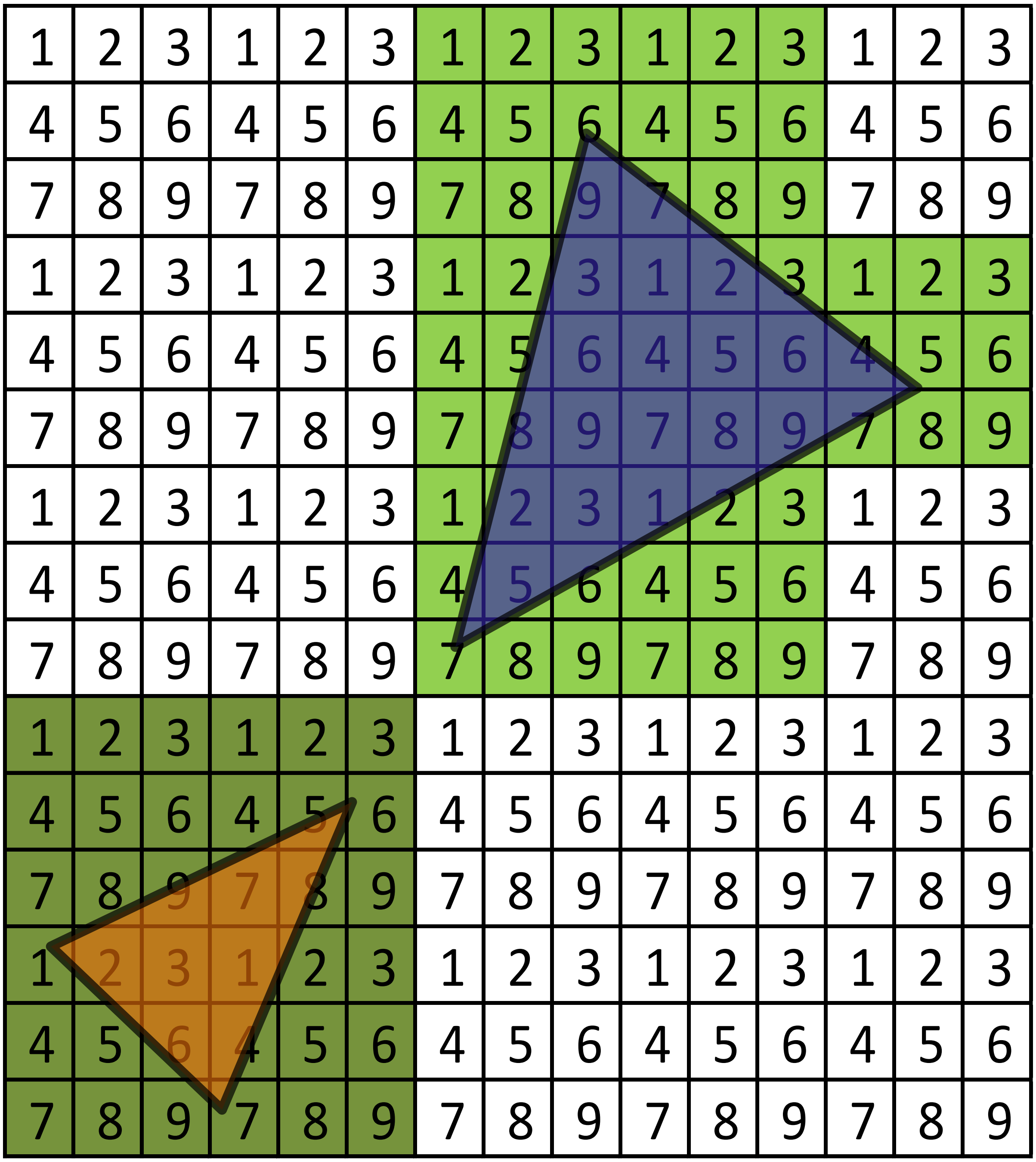


Figure 3.19: The cells affected by the scene adaptive balancing scheme, following a data modification for two differently sized view frusta

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 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 [48]. 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 (Figure 3.19). 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.

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.

Chapter 4

System Analysis

We will now analyze the proposed data management scheme on a 3×3 multi-projector display. Note, that since the technique proposed is for distributed data load balancing and data management, the technique will work on other configurations, regardless of its display technology. We have applied the technique to various configurations ranging from a 2×2 multi-projector display to a 5×5 multi-tiled lcd based display. 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 (Figure 3.5). Each SCD node consists of a 1024×768 Epson and Canon presentation projector, of 3500 lumens brightness, and an Intel i5 3.3GHz 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. [3, 51], 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. [49] 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. 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.

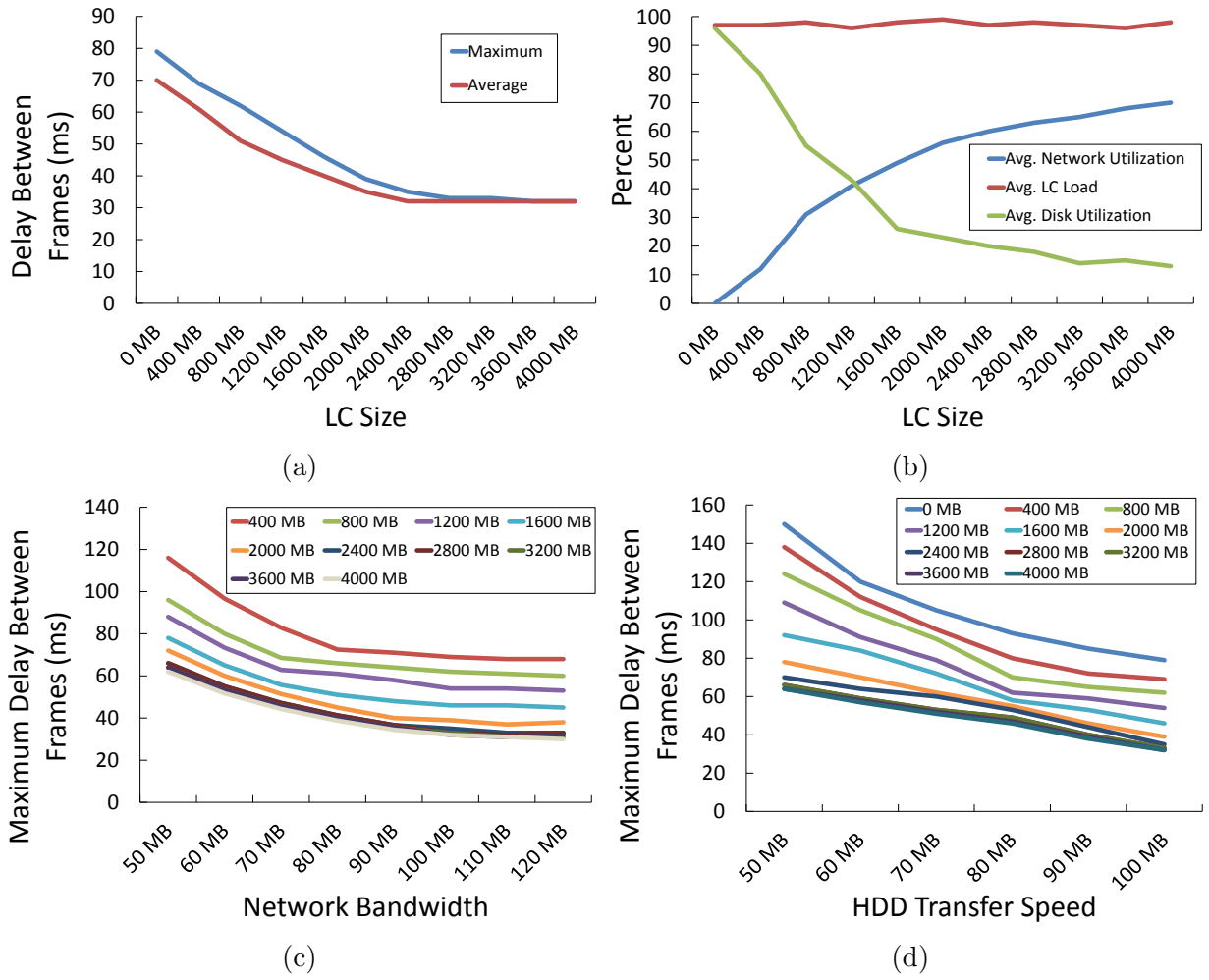


Figure 4.1: *System performance when c is set to the optimal value of 20 and f is set to 1. (a): Maximum and average delay between frames when system parameters are at their default values. (b): Resource utilization. (c): System performance with different network bandwidths. (d): System performance with different HDD transfer speeds.*

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. Figure 4.1 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 [54], 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 9 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 9 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. [49]), 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 previous section. 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.

Rendering vs. 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. Figure 3.17 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 Figure 24 of [16], 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 4.1, 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 4K 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.

Page Size	Compressed Size	Decompression Time
4K	1.8K	470 μs
8K	3.61K	811 μs
16K	7.18K	1411 μs
32K	14.3K	2296 μs
64K	28.5K	4351 μs

Table 4.1: *This table shows the compression size and decompression time, for different page size.*

Chapter 5

A Collaborative SAR Workspace for Modeling of Massive 3D Data

Today's trend is to get rid of personal offices and create large expansive work-areas where multiple individuals are placed in cubicles. The goal of such an environment is to foster a higher degree of collaboration and dialogue, which is not possible in an environment of closed door personal offices. Though, tools to interact with our personal data in our small personal world has exploded in the past few years, there are hardly any collaborative tools that can match the scale and modality of interactions that such work spaces can ideally trigger. There have been a growing body of CSCW and HCI work exploring the questions of creativity and collaboration to accomplish complex work, primarily in the area of technologies aiding collaboration methodologies [22, 11], yet all we see in such workspaces today are long stretches of white boards which hardly offer the digital tools and capabilities as we are used to in our personal environment.

We present a collaborative spatially augmented reality (SAR) workspace that is designed specifically for the evolving multi-user workspaces of today. This is a workspace where the

transition between personal work environment and shared work environment is minimized, thus enabling users to collaborate more comfortably. In this workspace, one does not need to huddle in front of a desktop, but has the freedom of using the huge amount of wall space (with or without whiteboard) available to him along with a plethora of interaction modalities (e.g. keyboard, mouse, laser pointer, gestures) to interact with other users and data. We show that the backbone of such a system can be achieved by utilizing multiple machines (maybe via clustering of all the machines used by the users in this workspace) to provide a very powerful computing environment. However, this work specifically focuses on how to use these machines in an efficient manner to provide a rich interactive and collaborative environment that is transparent to the user. This allows the user to use the displays as mediums of collaboration with the data and other users via any input modality suited for the particular situation and scenario. To demonstrate this capability, we use the example of collaborative modeling of 3D massive data. The rationale behind choosing this application is many fold.

1. A massive 3D model is impossible to interact with using a single computer. A backbone of a conglomeration of computers is essential to do anything collaborative or interactive with such models. This provides us an opportunity to justify using a complex computing backbone and demonstrate the critical capability of making it transparent to the user.

2. A 3D modeling environment provides us with enough visual and interactive complexity that would easily encourage multiple users to use the full capability of the collaborative SAR system that can provide a rich set of tools for interaction of and with the different number of displays.

3. Finally, today's 3D models are getting exponentially large and complex with no collaborative tools to match its scale and complexity so that multiple users can work easily

with such models. Therefore, there exists an opportunity for the methods proposed to be readily adopted in such applications.

5.1 Challenges in SAR Workspace

We present the first complete spatially augmented reality (SAR) workspace that prompts collaboration – a model workspace that can be easily deployed and would allow natural collaboration modalities. Users can create separate workspace to work on their own and create a shared workspace whenever collaboration is needed. We present a flexible distributed architecture that can be easily adapted to numerous visualization and collaboration needs by adding different modules to the base architecture.

Using modeling of massive 3D modeling as an example application, we demonstrate the following.

1. How a collaborative SAR workspace can be build over a complex multi-machine architecture?

2. How it can offer key capabilities of reconfiguring displays to multiple numbers, shapes and size to foster multi-group interaction in a large multi-user workspace?

3. How it can offer key capabilities of interacting with the data and other users via the displays (e.g. committing an edit operation or checking out a small 3D object from the entire massive model) and a large number of input modalities like lasers, gestures, keyboard and mouses?

4. How such a collaborative SAR environments can be used for many different collaborative needs?

5. Finally, we point out through a discussion how most aspects of this collaborative SAR system is easily generalized to other applications thus providing a rich set of tools for future researchers to develop such collaborative applications.

5.2 Related Work

Tiled multi-projector displays have been used in VR and visualization environments for a long time [13, 5]. However, they tend to be large inflexible systems driven by a centralized server and hence maintained by a crew of trained professionals. Different distributed rendering frameworks have been proposed to use them effectively for visualization of large data [44, 16, 26, 2, 29]. More recently, the work by Raskar et al.[41] proposes the idea of using immersive displays in office environments to create completely immersive 3D worlds where telepresence can be achieved in the truest sense. However, limited resources (e.g. network bandwidth) and imperfect methodologies (e.g. real-time 3D reconstruction of large scenes, holographic displays) has inhibited the progress of this grand vision. Our vision stands in between these two. We envision interactive high-resolution displays to be presented all around the user making them available to the user ubiquitously. Moreover, we envision them to be used as interaction medium to interact with data and other users in a shared collaborative workspace. We consider a distributed network of projector-camera systems mounted on pan-tilt units (PTU) connected to a backbone of PC clusters to create a new paradigm of collaborative spatially augmented reality (SAR) workspace to realize this vision.

Various form of augmented reality (AR) systems, where integration of virtual entities in the user's physical environments, have been proposed [31]. Single static projector-camera unit, lighting a large static surface (either wall mounted or on top of a table), has been explored to create multi-user interactive displays [27, 55]. The work by John Underkoffler [58], titled Luminous room, is a partially immersive spatially integrated environment that

uses a single projector to generate 2D images on static flat surfaces in a room to enhance the user's environment. In the domain of multiple projector-camera units illuminating a static planar surface creating large display walls, Bhasker et al presents a distributed network of projector-camera systems and associated distributed registration methodologies[3] that can help the user to repurpose the shape and size of a large display wall by changing the number of projector-camera units and their rectangular array configuration to adapt to different sizes and form factors of different visualizations (e.g. a 16:9 aspect ratio for a map visualization and 4:3 aspect ratio for biological visualization). Roman et al presents a distributed interaction paradigm for interacting with 2D applications (e.g. multi-user graffiti, emergency management, digital bulletin board, interactive collaborative map annotations) on such a display formed by a network of projector camera systems[46]. Some systems have merged synthetic images with real scenarios for a static user or view point. The work by Dorsey et al provides a useful framework in the realm of theater set design [15]. In this work, a pre-distorted image from multiple devices appears correct when projected onto a static curved backdrop of the theater. Finally, Illumiroom from Microsoft uses multiple projector-kinnect units to generate images on the static 3D world (e.g. furnitures) surrounding the gaming console (usually a TV) to create a greater sense of immersion. Note that 3D cameras are used here instead of 2D cameras in all the other aforementioned works.

The concept of projecting using a single dynamic projector-camera system on static environments has also been explored in the past. [38] created the first wall or ceiling mounted steerable display using a single projector-camera unit. This unit augmented interfaces on static, mostly planar, 2D objects/environments [37] to providing a novel interaction modality. [40] presented multiple uses of hand-held single projector systems augmenting real world physical objects and [8, 7] showed how multiple users can interact with each other on a large virtual space using their personal mobile hand held projector-camera units. The work by Sueishi et al [56] explores the use of a motorized mirror to move the display output of a single projector around to light moving objects. This work focuses on tracking moving

objects in the real world and orient the mirror such that the object is always illuminated by the projector.

For this work, we still operate in the domain of a static environment and demonstrate the power of having multiple dynamic (or steerable) projector-camera units. Further, we bring in the idea of using such dynamic units to dynamically change the association between the users personal workspace and a shared collaborative workspace. This flexible reassociation of different devices brings in an hitherto unseen flexibility for users to segment or merge personal displays from and to shared displays respectively. This enables the creation of a novel collaborative SAR workspace where a plethora of dynamic display units are easily available as a resource to connect these displays personally or in a group and interact with the data and other users (local or remote) through the available displays.

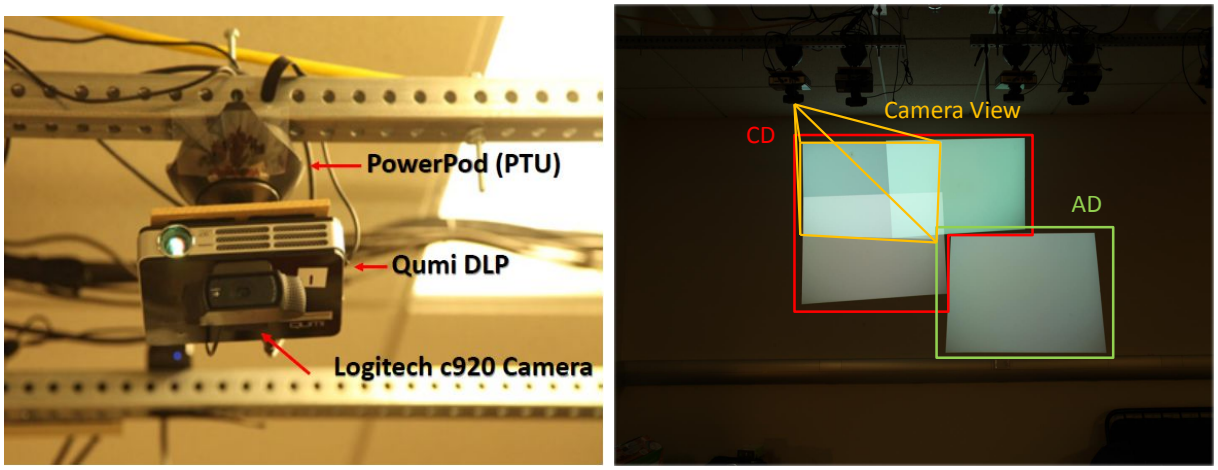


Figure 5.1: Left: This shows an active display node (ADN) with the projector, camera and a pan-tilt unit (PTU). Right: The projection of the ADN here is indicated by the green boundary. A conglomeration of ADNs called a conglomerate display (CD), is indicated by the red boundary. The orange view frustum indicates what an ADN's camera would see on the wall. Note that the ADN's camera field of view is larger than that of its projector as is common in commodity devices.

5.3 System Overview

Our collaborative spatially augmented reality (SAR) system consists of a conglomeration of active display nodes (ADNs) mounted on the ceiling of the workspace. Each ADN consists of a projector and a camera, mounted on to a pan-tilt-unit (PTU). All the ADNs are then mounted on rails that extend from the ceiling of an office (Figure 5.1 and 5.2). Each ADN is connected to a computer (can be the PC of one of the users in the shared workspace) and more than one ADN can connect to the same computer. This conglomeration of ADNs are connected via a network. Therefore the backbone architecture of collaborative SAR is very simple, a conglomeration of ADNs driven by a PC cluster (Figure 5.3). Each ADN is capable of movement, which is achieved by the PTU. The projector that is mounted on the PTU is analogous to a flashlight in a person's hand. The PTU can orient itself in any configuration thereby pointing the projector at any place in the office. Thus a display can be placed anywhere in the workspace to create a display space. When an ADN is positioned contiguous to others, the system can create a larger seamless display by aggregating all spatially contiguous ADNs, effectively increasing the size and resolution of the display. We call this grouping of ADNs a conglomerate display or CD (Figure 5.1). With this capability, users are able to move the ADNs around to form smaller individual workspace or to join the ADNs together to create one or more CDs of different size and form factor. Our collaborative SAR workspace is shown in Figure 5.2.

5.3.1 Interaction for Display Reconfiguration

Displays in personal computing environments (e.g. PC, tablet, phone) today are becoming of such small form factors that they cannot be used in an ergonomically comfortable and functionally efficient manner for long collaborative sessions. On the other hand, people are comfortable sharing a larger physical space (e.g. a meeting room, or a table surrounded

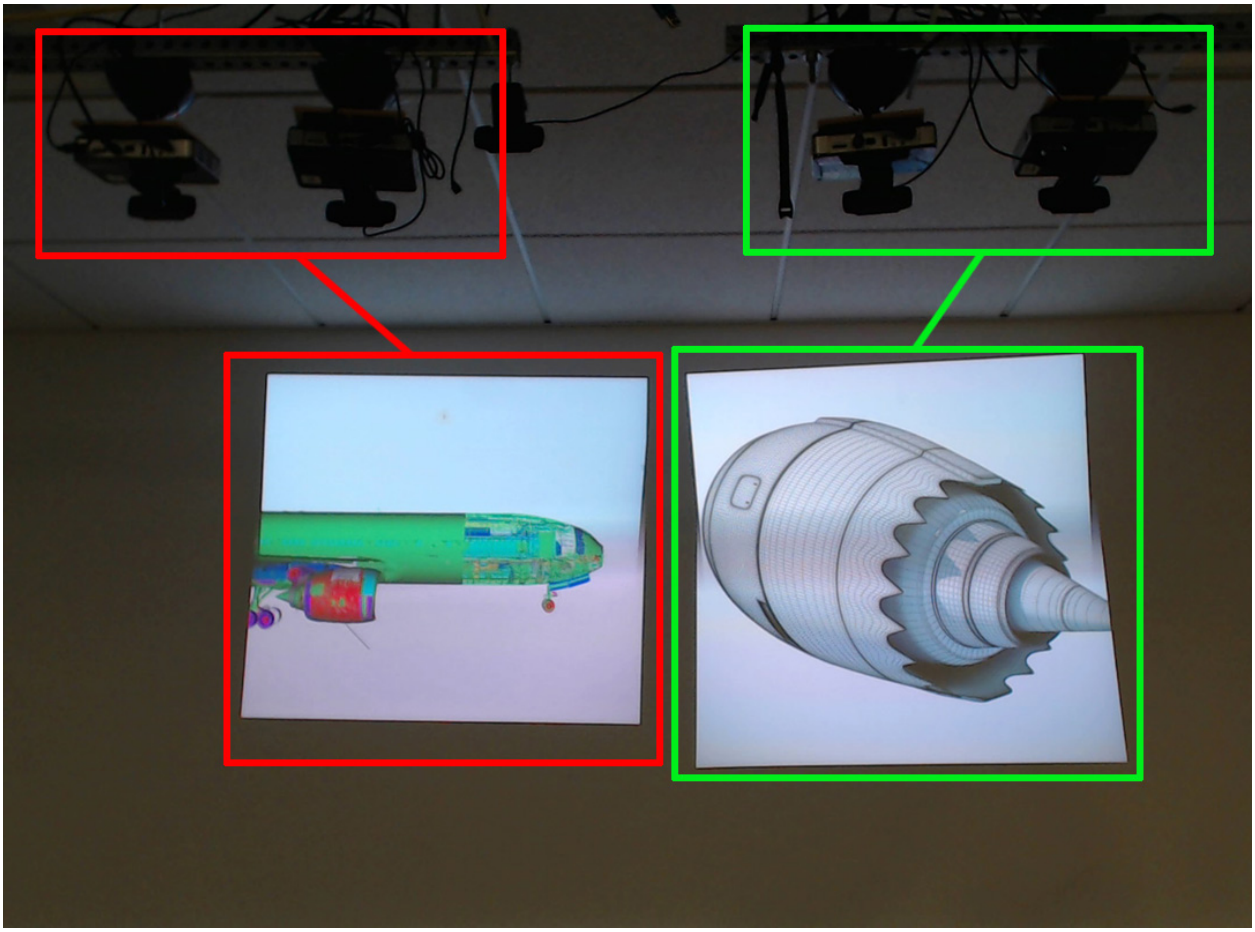


Figure 5.2: This shows our SAR collaborative workspace made of four ceiling mounted ADNs creating two CDs each with two ADNs.

by chairs in a big workspace) for collaborative dialogue and discussions using devices of much larger form factor like white boards and smart boards. So, it is imperative that if the users are provided with the capability of bringing such a larger form factor display anywhere in their shared physical space, they will be encouraged to use this for collaborative purposes. We therefore provide interaction capabilities by which the user can move the ADNs using gesture or a laser, and connect them together in different configurations by a simple mechanism of placing them in a spatially contiguous fashion. These different configurations include changing the number, position and configuration of the ADNs forming the CDs that changes their size, resolution and form factors accordingly. This allows users to have a set of displays that can change over time based on the specific need of their application.

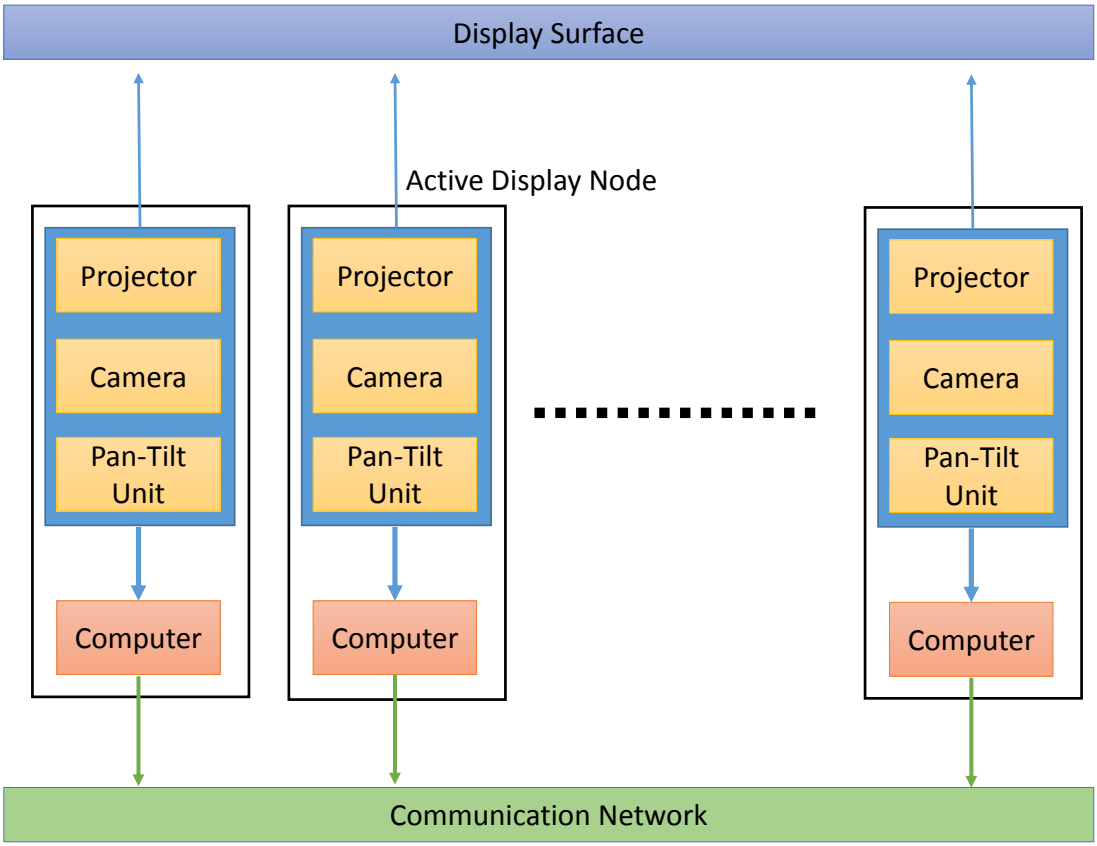


Figure 5.3: Our architecture is a general one that consists of each ADN connected to a computer which are connected by a communication network.

For example, a few users may be using a large display of 4K resolution to visualize a large 3D model. When they each want to check out a different part of the model for editing, they can segment the display into three: (a) one of 2K that still shows the whole model; and (b) two other displays, each of 1K size can be segmented away from the 4K display to create two smaller environments for the users to edit the respective model parts they have checked out. Once they are done with editing, merging the personal 1K displays with the 2K shared workspace commits the changes to the main model and brings back the 4K shared display. Such interactions are only possible when the user has extreme flexibility to reconfigure the multiple ADNs into different number of differently sized and shaped displays

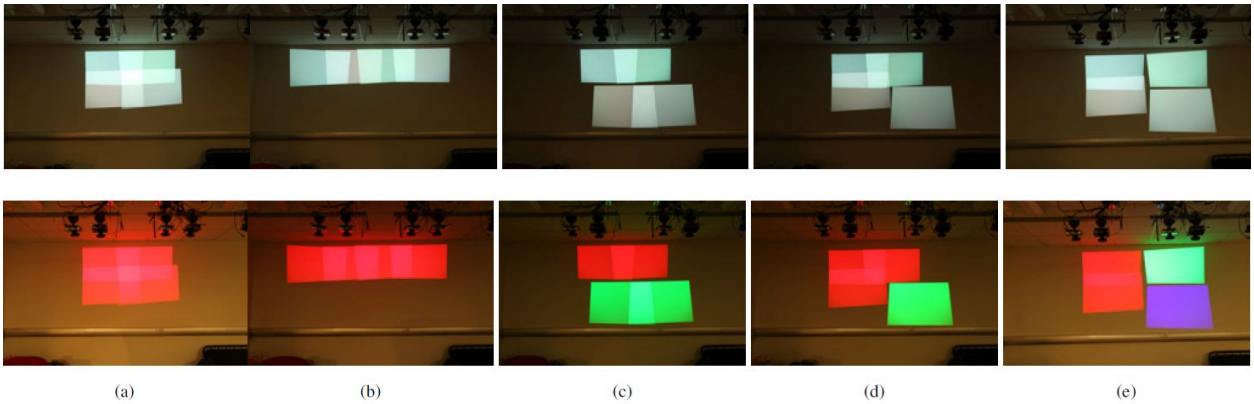


Figure 5.4: The top row shows the ADN placement before registration. The bottom row shows the grouping of the ADNs to create the CDs, with various configurations. Some examples of such configurations are: (a) one CD, made up of 2x2 projectors; (b) one panoramic CD, made up of 1x4 projectors; (c) two panoramic CDs, each made up of 1x2 projectors; (d) one CD using 3 projectors in a non-rectangular fashion, and another CD made up of a single projector; (e) one CD made up of 2x1 projectors, and two CDs made of a single ADN each.

as shown in Figure 5.4. The different colors indicate the system identifying the different CDs by their conglomeration via spatial contiguity. This is what we call *interactions for display reconfiguration.*

This interaction is enabled by the camera on each ADN that provide visual input to trigger movements of the PTU to move the display around. Users can use a laser pointer or hand gesture to achieve the movement and hence the reconfiguration. To alleviate the user from situations of making complex decision like (a) are the displays connected or segmented with or from each other; or (b)do they have enough overlap to create a nice seamless display; the system provides intuitive visual feedback. When the user selects an ADN to move, it is highlighted as white. When the user starts moving an ADN, the boundaries of all other ADNs are highlighted with red. As the user moves the ADN, its white projection allows tracking of the ADN. Once the ADN enters another ADN, it can compute its overlap with the moving ADN well defined by its red highlight. Once this overlap is beyond a threshold, the boundary of this ADN turns green to indicate to the user that enough overlap has been

achieved (Figure 5.5). Once the user has reached the desired configuration, the system runs an automated registration to identify and register the imagery coming from multiple ADN units into one or more seamless CDs. Please see the video to see these interactions in action.

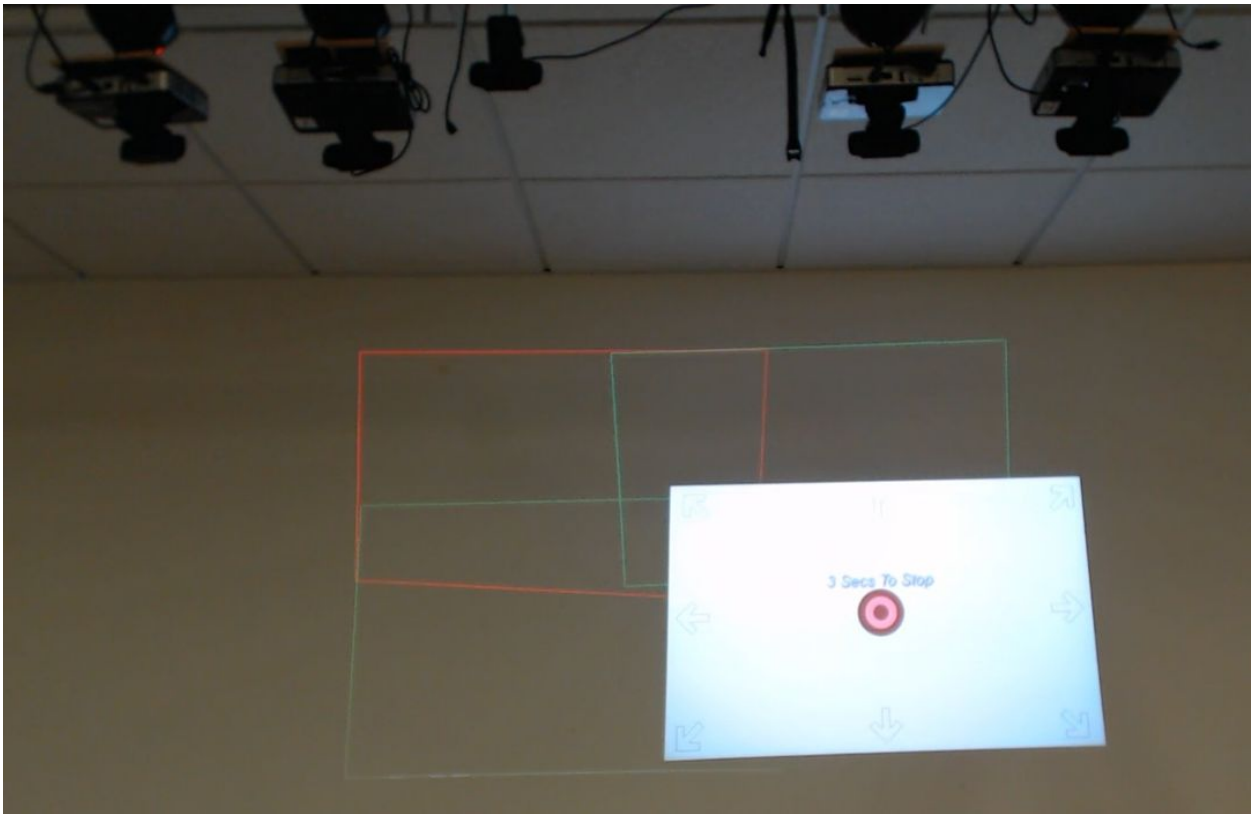


Figure 5.5: This shows the placement feedback given by the ADNs to assure that the users have enough overlaps when creating their CDs. The moving ADN is highlighted in white. The ADNs of the CD with which it is merging is turned green to say that the overlap is adequate to create a seamless display. Note that one of the ADNs are still red indicating not enough overlap with it. To merge with this display, that user has to make sure that all the ADNs in the CD that overlaps with the moving ADN should be turned green.

5.3.2 Interaction for 3D Manipulation

The first capability of a collaborative system is to allow users to connect to the shared display easily. We have developed an application by which one or more users can connect to a CD and port the whole or part of their desktop on the CD. The position of their desktops on the

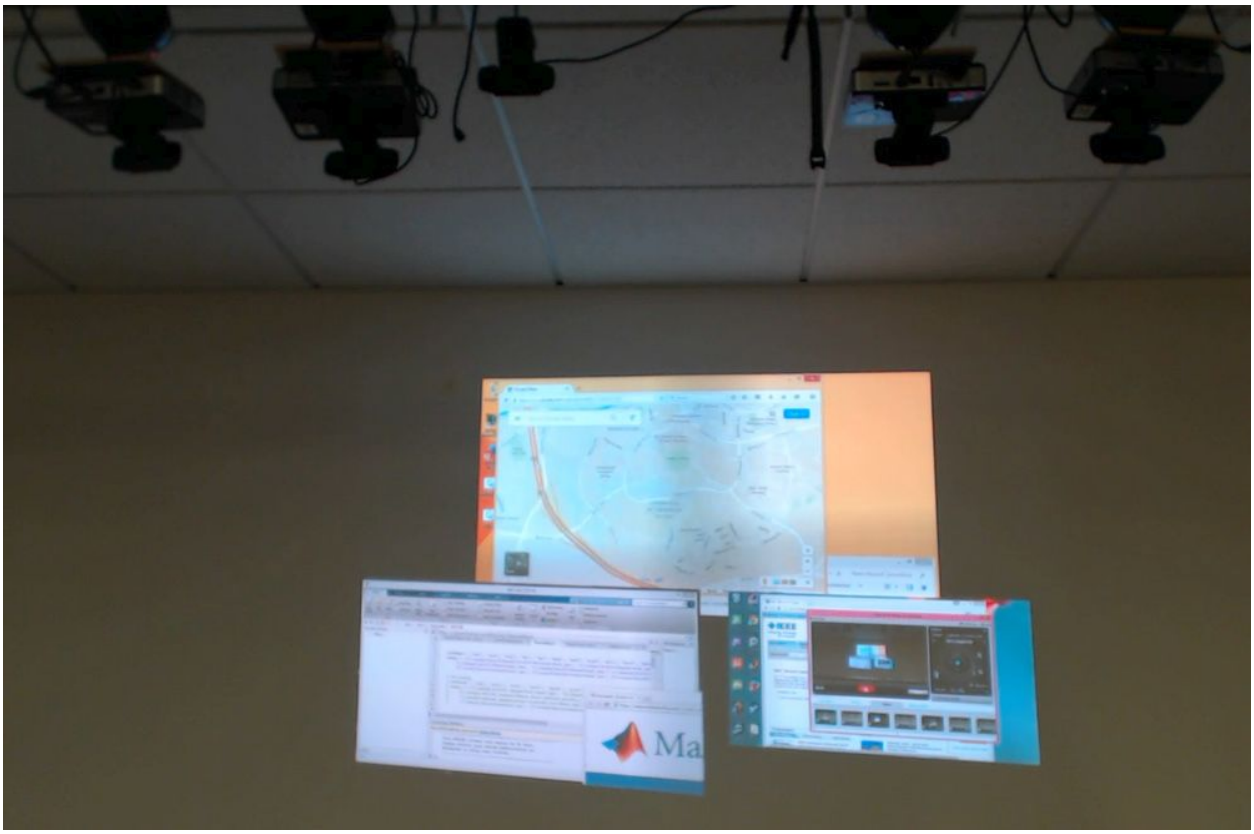


Figure 5.6: This shows three different desktops created using our collaborative SAR framework.

conglomerate display can be easily moved around using a laser pointer or a control interface. This is illustrated in Figure 5.6 and 1.

In a walkthrough 3D environment, the user can use various types of inputs to navigate through the environment. We have implemented a laser pointer based interaction mode defining three types of movement: forward/backward, rotate left/right, and pan up/down, and tied them to different laser-based gestures. To move forward/backward, the user creates a straight line from the bottom to the top portion of the CD and vice versa. To rotate left/right, the user creates a straight line from the left to the right portion of the CD and vice versa. To pan up/down, the user creates a diagonal line, where a line direction between 20 degree and 70 degree is interpreted as a pan up and a line direction between 200 and 250 degree is interpreted as a pan down. These movement operations would allow the user

to fully explore the 3D data set. However, such interactions can be designed for this and any other application as per the users' specifications using the interface modality (e.g. hand gestures instead of laser based gestures) he chooses. Please see the video to see these in action [28].

We also allow a display driven data operations like data check-out, edit and commit step for shared 3D data manipulation to avoid data inconsistencies. The user can trigger checking out of a part (e.g. engine of a plane) of a bigger 3D model being visualized collaboratively via a few blinks from the laser pointer. This part then can be moved to a smaller part of the display which can be segmented out from the bigger display using the same mechanism for repositioning the AD thus allowing the user to edit the part of the model on his own without disturbing the shared model visualization. Once done with his edits, the user moves the segmented display back towards the bigger display triggering both a merging of the displays and a commit to apply his changes in the data which shows up in the shared collaborative visualization. This is illustrated in Figure 5.7. The same paradigm allows the user to treat his own desktop display as a extension of his shared CD and make changes there which are reflected in the wall-top CD as shown in Figure 5.8.

5.4 Front End Processes

Several front end processes work together to achieve the interaction for display configuration and for 3D data manipulation. In this section we describe them in details.

5.4.1 Interaction for Display Reconfiguration

We consider the ADN's as a set of active agents who are working together to create the collaborative SAR. Therefore, we develop completely distributed methodologies for grouping

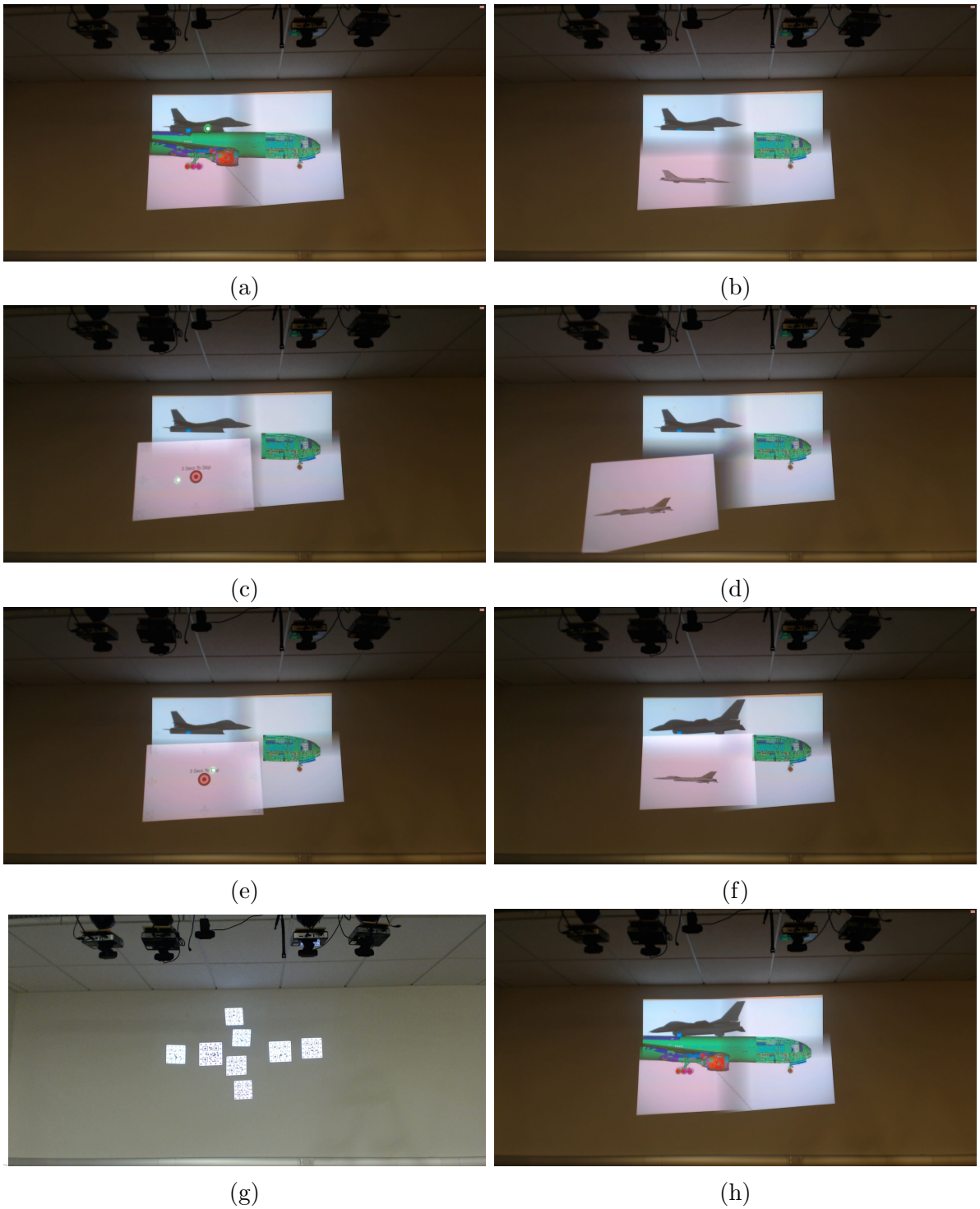


Figure 5.7: *(a) The user selects the F-16 using a laser pointer and selects the lower left screen as the destination to check out the model. (b) The F-16 model is now in a separate context on the lower left screen. (c) The user uses ADN reconfiguration mechanism to move it away – segmented from the display. (d) The model can now be edited without affecting the original data set. Here the model has been changed by rotating it 180 degrees. (e) The user moves the personal ADN back into the shared larger CD. (f) The changes to the model are now committed to the original data set. (g) Calibration is performed. (h) The calibrated CD with the latest changes.*

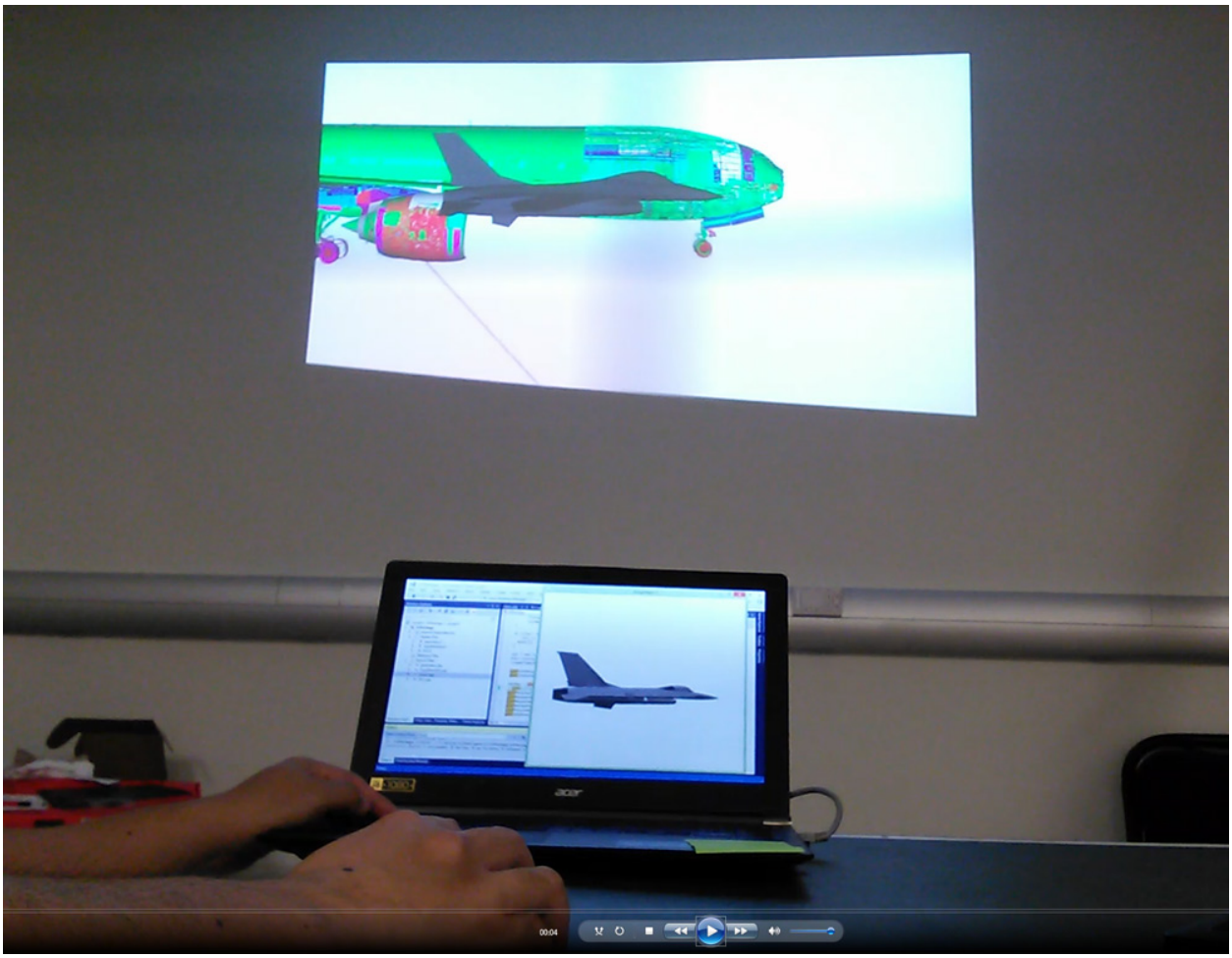


Figure 5.8: A desktop extension of the CD. Left: The user starts with the model on the desktop and performs a scaling. (b) The desktop changes and the change is reflected on the wall-top CD.

and regrouping of the ADN's to create the conglomerate displays (CDs). This allows easy addition and removal of the ADNs to the collaborative SAR. The output for this process is a configuration file for each ADN, that any application can use to understand the configuration of the CD. Each configuration file contains its IP address, a list of the ADN's neighbors and their IP addresses, and the geometric transformation to warp the 2D image from the ADN into the display space. With each reconfiguration, the configuration file of the affected ADN is changed. An API is available for developers to create their own applications that can be

integrated with the system. A user can query the system for the configuration information from each ADN.

When the system starts up, the position, orientation and ID of each ADN is unknown. In fact, each ADN does not even know the number of total ADNs in the system. The ADNs have to go through a process of making them known to the system. For this purpose we plan to use the distributed registration technique used in [46] for registering a single tiled display made of a rectangular array of projector-camera units projecting on a planar display. An SPMD (single program multiple data) algorithm runs on each unit that starts with the assumption that it is the only unit in the environment. Then it performs a *configuration identification* step that goes on to discover all the other units, its own location and the location of its neighbors in the rectangular array. Finally, in a *registration* step a SPMD method achieves seamless registration of all the images from the multiple units. This method assumes rectangular overlaps of roughly fixed widths and achieves the configuration identification and registration via QR codes placed on these similarly sized overlaps.

Though the algorithm in [46] extends well to our scenario of multiple CDs, our collaborative SAR system is still significantly different than the system in [46] due to the following reasons. (a) Since we deal with projections from ceilings, they show considerable keystoning that cannot assure rectangular overlaps. (b) Since we give the user complete freedom to overlap ADNs with each other as they please, neither can we assure similar sized overlaps nor can we assure overlaps to be strictly to the left, right, top and bottom of each unit. (c) Finally, if the user does not provide adequate overlap, the system needs to guide the user to provide adequate overlap. To allow for these additional flexibilities we have added three steps before the configuration identification and registration steps in [46] can take over. These steps for each ADN are (a) Overlap Discovery; (b) Placement Feedback; (b) Conflict Free QR code placement. We assume that the camera field of view (FOV) in an ADN is reasonably larger

than the projector FOV. This assumption is derived from the common FOVs available in commodity cameras and projectors.

Overlap Discovery

The goal of this step is for each ADN to identify the overlap regions around its boundary, their size and shape so that it can achieve a conflict-free QR code placement. For this, as soon as each ADN is powered on, it projects a white image and observes it. Note that if none of the ADN's overlapping neighbors project at the same time, the larger field of view of the ADN would be able to see its one white display surrounded by an dark or dimly lit (if some ambient light is present) region in all directions. Further, when any of its neighbors projects the white, it should be able to observe it through the larger field of view camera to detect the overlap, its shape and size. It assigns the overlap to be a left, right, top or bottom one (as is required by [46]) based on the largest number of pixels present in its four quadrants. For e.g. If the overlap has more pixels in its top right than bottom right quadrant, it is assigned as a overlap on the right of the ADN. In order to handle conflicting projections from multiple ADNs at the same time we propose an algorithm akin to the Carrier Sense Multiple Access/Collision Detection (CSMA/CD) method in the network domain. In this, any ADN trying to project at any time first senses if a neighboring ADN is projecting via the visual input from the camera. This is easily detected by the a white area which shares the boundary of the ADN. The ADN projects only if another ADN is not projecting in its area. Otherwise, it waits a random amount of time and retries. This continues until all ADNs construct their overlap areas and label them as left, right, top or bottom. Finally, each ADN computes a rough homography that relates itself to the different ADNs having overlaps with it. This is achieved as a two step process. First the $i$the ADN discovers the four corners of its own projection in its camera space to find the homography $H_{P_i \mapsto C_i}$ from its projector space to the camera space. Next, it finds the four corners of the overlap with a neighbor j to find the

homography between the camera in the ith ADN and the projector in the jth ADN, given by $H_{C_i \mapsto C_j}$. These two homographies are then concatenated to provide an homography between the projectors of the ith and jth ADNs, i.e $H_{P_i \mapsto P_j} = H_{P_i \mapsto C_i} H_{C_i \mapsto C_j}$. These homographies are approximate since they are computed using only four correspondences.

Placement Feedback

For interaction for display manipulation, we need to provide a mechanism for the user to move displays around. This can be done by a laser-based or hand gesture (open palm). Once the ADNs are powered ON, the user can use gestures to position them the way he wants to create the CDs. Every ADN has a designated hot-spot-switch area. If a gesture is detected in this area, the ADN switches to reconfiguration mode. The user starts with a gesture in this switch-hot-spot area of the ADN he desires to move. The ADN turns white to indicate its status of being chosen to be moved. A broadcast message lets all the other ADNs know of the existance of a moving one and they turn their boundaries red. When the user moves the selected ADN, the white projection allows continuous tracking of the moving ADN via updated $H_{P_i \mapsto C_i}$. As this moving ADN enters the field of view of any other ADN, it can track the overlap easily due to the red boundary and computes the amount of overlap in the projector space using $H_{P_i \mapsto C_i}$. If this overlap is above a threshold, the observing ADN turns it boundary green to indicate enough overlap. Therefore, user has to make sure that whenever he is merging, all the ADNs he is merging with should turn green. While segmenting the display, visual feedback is of less use since all the user needs to do is to make the moving ADN spatially disconnected from the existing CD. Once the user finishes moving the ADN, another gesture in the same switch-hot-spot area moves him out of the reconfiguration mode. Since relative position of the projector and the camera in a single ADN does not change with movement, the location of the hotspot remains the same across this movement.

Note that for moving the ADNs intuitively, the projected area from the ADN should move along with the user gesture. This demands a fast and simple interaction recognition and tracking. For laser based interaction, we use a simple and fast image processing to detect the laser highlight and move the ADN quickly with it. The steps of this method are as follows. (a) Point the laser to switch-hot-spot area to switch mode; (b) a red circle appears on each ADN; (c) select the ADN by holding the laser on this red circle of the desired ADN; (d) the selected ADN turns white and the other ADNs display red boundaries; (e) move the ADN using the laser and stops the movement when the ADN(s) with which it is merging turns their boundaries green indicating sufficient overlap to create a seamless CD; (f) point to the switch hot-spot to switch out of the reconfiguration mode.

However, note that achieving the movement of the ADN with the hand gesture is computationally very demanding since gesture detection is still not real-time process. Since the user is not engaged in any work on the displays during this phase of configuring the conglomeration, we use the hotspot-based gesture recognition technique proposed by Chiu [10] for this purpose. This greatly increases the accuracy of gesture recognition at a very low latency allowing the ADN to move along with the user. The steps of this method are as follows. (a) Change from display mode to reconfiguration mode by placing palm on switch-hot-spot area. (b) Project blob pattern in the display space. (c) Identify the open hand gesture used to select the desired active display. (d) Track the movement using hotspot-based tracking and move the active display to a different region. (e) If there are no movement for more than 3 seconds, identify that as the culmination of the reposition operation, and deselect the active display switching out of reconfiguration mode. This is illustrated in Figure 5.9.

Conflict-Free QR Code Placement

In [46] the registration is achieved by first presenting a QR code by each projector in their overlap region that encodes its IP address, port number, and ID into the QR code. Note

that the overlaps are all trapezoidal due to keystoning and can be differently shaped and sized. This leads to two issues. (a) A rectangular QR code in the projector coordinate system results in a trapezoidal QR code on the display which cannot be detected using standard QR code detectors due to severe resolution compression in some of its regions. (b) Second, a standard placement of QR codes as in [46] can overlap or conflict with the QR code from another ADN. So, we introduce two steps to alleviate this situation. First, we use the approximate homography $H_{P_i \mapsto C_i}$ computed in the previous step to apply a pre-warp to the QR codes so that when projected it looks rectangular. Second, we choose the size and placement of the QR code using the SPMD method shown by Algorithm 2 to achieve a conflict-free placement. The effect of this algorithm is shown in Figure 5.10.

Algorithm 2 Generate QR Code per ADN

for all ADN **do**
H = homography from projector to display.
KT = keystone correction transformation calculated from H
OL = Number of overlaps with other ADN.
for all OL **do**
If (Two ADNs overlap an area (this ADN and a neighbor ADN))
{Find maximum rectangular overlap area. Mark area as safe.} Else if (More than two ADNs overlap an area)
{Mark area as hazard.}
end for
for all Safe Area **do**
Generate QR code, maximize size with consideration to overlap s.t. the size is less than half of the safe area's longest dimension
P = Determine position of the overlap (Top, Bottom, Left, Right)
If (P == Top {Place QR code with a weight toward the left edge of the safe area}
Else If (P == Bottom {Place QR code with a weight toward the right edge of the safe area}
If (P == Left {Place QR code with a weight toward the bottom edge of the safe area}

If (P == Right {Place QR code with a weight toward the top edge of the safe area}
end for
I = Image containing all placed QR code
CI = Apply KT to I
CI is used in the registration phase
end for

Once the QR codes are detected, each set of ADNs that are spatially contiguous form a CD and get to know each other's IPs to talk to each other via the network communication. They use this dialogue to find the ADN with the largest number of neighbors using the largest amount of overlapping pixels as a tie breaker to decide on the reference ADN for each CD. Once the reference is decided, the cascading homography method in [46] takes over to label them, find their configuration and achieve seamless registration. Note that though the entire system can have n ADNs, the labels of ADNs in each CD will be no more than m which is the number of projectors present in that specific CD. Since multiple CDs can exist, ADNs in different CDs can have the same label. The application querying the system finds out the number of CDs in the display. A CD before and after registration is shown in Figure 5.11.

5.4.2 Interaction for 3D Model Manipulation

Once the CDs are created, and the personal displays are connected to a CD (described in Section 3.1), gesture based interactions are used to navigate or edit the models in collaboration or alone. We use laser based interactions since they seem to be ergonomically most appropriate when dealing with large displays and people are quite used to using distal interaction devices like remotes and laser pointers today. The gesture data is composted of a list of 2D points, that represents the position of the laser in the CD. Further, unlike hand gestures recognition, which are susceptible to environmental lighting conditions (e.g. when content is projected onto the hand) laser is more robust and resilient to environmental conditions. Since the light intensity of a laser pointer is very high, it is very easy to threshold the visual input to accurately find the laser point. This can be done regardless of the content that is being projected. Furthermore, different laser colors can be used to denote multiple users in a collaborative environment.

Note that it is possible for a gesture (panning by movement of laser) to span multiple ADN in a CD where each ADN sees only a portion of the gesture. Or, it is possible for an ADN not to see any gesture at all, but be expected to react to the gesture (when the gesture is confined to a part of the CD not seen by an ADN). To assure appropriate hand-off of gestures and handling of race conditions, we adopt the distributed interaction paradigm proposed in [46]. Each ADN runs the distributed SPMD (single program multiple data) gesture management and reaction management technique presented in [46]. With each gesture, the CD reacts using the backend processes described in the next section. Check the video to see these interactions live. The gesture information is also provided through the API. The user can use the gesture information to define new gestures specifically for their application.

5.5 Back End Processes

In this section we describe all the back end processes that makes collaborative SAR for massive 3D modeling a reality.

5.5.1 Data Management

A major bottleneck in modeling of very large models that cannot be instantly fit into the RAM of a single machine is the data management without creating duplicates so that data consistency can be maintained easily. [29] uses the same architecture as shown in Figure 5.3 in a sort-first rendering architecture to propose an interleaved data partitioning along with associated methodologies that can achieve a load balanced (both in terms of storage and rendering) data management using a PC cluster in the back end. We use this data management technique to achieve modeling of massive 3D data. This method assumes a static number of machines in the PC cluster and proposes a data partitioning preprocessing

which is then adaptively repartitioned in real-time with any edits that add, delete or move data during runtime. Figure 5.12 shows a 20GB Boeing model being partitioned on a 9 PC cluster using this method.

In our work, since the conglomerate display (CD) on which the data is shown can have multiple configuration during a work session, it is difficult to reprocess the data for partitioning on a different subset of machines every time a reconfiguration happens. To alleviate this problem, we consider the data backbone of the system to be comprising of all the ADNs, i.e. all the PCs present in the shared workspace. This allows us to achieve the display reconfiguration on a subset of machines only in the front end for the purpose of user interface and shared processing the rendering load, while all the machines in the workspace share the load of data management in the back end. This assures that data requests coming from any CD is guaranteed to be available in the system. Having a larger number of PCs in the back end also assures better performance in terms of data access and management. To accommodate any reconfiguration, we use the calibration data to identify who are the neighbors of each ADN with respect to their current position to create a lookup table that compensates for the change of neighborhood information that was calculated during the preprocessing phase. Figure 5.13 shows the comparison of 20GB Boeing model being partitioned to the subset of two ADNs or all the four ADNs when creating a 2x1 CD. Note that the load is much better balanced for the latter which increases performance as observed in [29].

We also compared the performance in maximum frame delay (measures the maximum stall that can be faced by the user) and average frame rate (measures the average interactivity) of the Boeing model rendered by the subset of ADNs which form the CDs versus having more ADNs involved in the back end data management. The results are presented in Tables 5.1 and 5.2. The rows indicate different display configurations: (a) a single display (1x1); (b) the whole display including all 4 ADNs (2x2); (c) 1 CD made of a 2x1 ADNs; and (d) 2 CDs each made of a 2x1 ADNs. The columns indicate the number of ADNs involved in the

data management in the back end. The emboldened numbers are the configurations where the number of ADNs rendering matches the number of ADNs managing the data in the back end – the configuration proposed in [29]. The other numbers indicate the performance of configurations enabled by our extension of the work. Since by definition, the number of data management nodes should be at least as much as the rendering nodes, the configurations where the rendering nodes are smaller than the data management nodes are not applicable.

Some interesting observations can be made from these tables. First note that for a single ADN rendering, when the data management is done on multiple ADNs instead of one, the maximum frame delay increases. This indicates that the overhead for data processing is higher than the benefits provided by the partitioning. However, the average frame rate remains almost the same indicating that this impacts the performance only in the worst case. Further, since the performance is already very poor (about 1 fps), this impact is probably inconsequential. The more interesting data is offered by the third and fourth row which shows that when more machines are involved in the back end, performance improves both in terms of the stall faced by the user and the average frame rate experienced, being the best when all the ADNs are used in the back end. Finally, when using two CDs from 4 ADNs versus 1CD from 4 ADNs, the latter shows almost double the performance in terms of frame rate and marked improvement in terms of maximum frame delay. This can be attributed to the use of the cache of spatially adjacent ADNs, termed as the adjacency cache in [29]. The adjacency cache is much better utilized in the latter due to the spatial coherence offered by a single CD and therefore the superior performance. Note that our testbed is a small system with just 4 ADNs. As analysis shows in [29] performance goes up faster as the number machine are increased further.

Table 5.1: Maximum Frame Delay in milliseconds

Configuration	1 ADN	2 ADN	3 ADN	4 ADN
1 x 1	**1656**	2008	2255	2380
2 x 2	na	na	na	**85**
1 CD (2 x 1)	na	**160**	144	119
2 CDs (2 x 1 each)	na	na	na	**125**

Table 5.2: Average Framerate in frames per second

Configuration	1 ADN	2 ADN	3 ADN	4 ADN
1 x 1	**1.0**	1.0	0.9	0.9
2 x 2	na	na	na	**25.0**
1 CD (2 x 1 each)	na	**11.0**	12.0	14.0
2 CDs (2 x 1 each)	na	na	na	**13.0**

5.5.2 Desktop connection to a CD

One of the critical capability in our collaborative SAR environment is for a desktop to connect to a CD so that the user can run any application from his desktop on the wall-top shared display. In order to achieve this each ADN has a dedicated channel to accept external image data. A simple protocol is used to send image data to the ADN. In order to communicate with the ADN, a device needs to be on the communication network. Using the configuration information, a client can identify each ADN and send the corrected image data. For receiving image data, the system can operate in two modes: client-centric mode, and ADN-centric mode. In client centric mode, the client uses the configuration information to partition and correct the image for each ADN and send each ADN their respective image data according to their display space. Since the separation and image correction can be computationally expensive for a device, when the number of ADN is large, the system can also accept full image data, this is called ADN-centric mode. In this mode, the client sends the entire image data to all ADN in the CD. Each ADN will partition the image data and transform the image to the display space. Since the complete image data is sent to all ADN in the CD, each ADN will receive data that it doesn't need. Therefore, the network will be saturated with duplicated data. This will severely limit the number of video streams.

So, we tend to operate our system in the client-centric mode unless we have a very fat data bandwidth. However, as the network capacities in workspaces continue to increase, this may turn out to be an non-issue.

5.5.3 Multiple Desktops on a CD

To stream multiple desktops, each desktop needs to have a client. The ADN separates the number of sources based on the originating host address. Each source will be allocated an image data buffer to store the incoming data and make it available to the renderer. Using the configuration information, each client identifies where it wants its content to displayed on the CD. The client captures its desktop and sends the partitioned and corrected image data to the appropriate ADN in the CD. The same operation is used when a user wants change his desktop position or size on the CD. As described in the previous subsection, the system can operate in two modes in this case too.

5.5.4 Interacting with Data Via Display

We introduce a new modality for interacting with data by rearranging the ADNs. One of the fundamental challenge when sharing resources is version mismatch, where multiple users work on outdated version of a resource and hence conflicts arises when the resources are committed back into the repository. Since this system enables users to collaborate and work in a shared environment, we propose a version control mechanism using the display. The mechanism supports two operations: checkout and commit. When a resource is checked out, it is locked and no one else will be able to checkout that resource. Once a resource is committed, the resource is unlocked and is available for checking out. We demonstrated this mechanism on a 3D model.

As per [29] when the initial data distribution is done, a meta file is created which is then shared with every ADN. This meta-file contains the information of each data block and in which ADN each data block is stored and where. This is file that is updated during runtime redistribution. To implement the data checkout, the specified ADN(s) forming the personal CD will take over the object, as indicated by the user, and create a local copy of the object rendering only that object. When a user selects a portion of the model, the system will group all the data blocks into an object in the model. Each ADN in the CD will mark the objects data block as locked in the meta-file which is then broadcast to other ADNs as well. The personal CD switches context to an independent CD, and no longer associates itself original shared CD. Note, that this effectively removes the personal CD from being neighbors with the other ADNs in the shared CD and therefore, the adjacency cache of this CD is no longer available. Once this dissociation is achieved, the user needs to reposition this personal CD such that it does not overlap with the shared CD – essentially the personal CD segments out from the shared CD. The user can modify the local copy of the object in the personal CD and it will not affect the original data due to the lock on the metafile. Data redistribution due to the edits will be limited to the personal CD. When the user wants to commit the changes back into the original data set, the user repositions the personal CD to overlap with the shared CD. Through the visual input, the shared CD can detect an intention to join from another CD. The system recalibrates itself and the personal CD containing the modified data will unlock the data in the meta file which is then broadcast to all the ADNs. Once this is achieved, the next redistribution triggered by the edit will use all the ADNs to achieve the redistribution and hence a better load balancing of rendering and data.

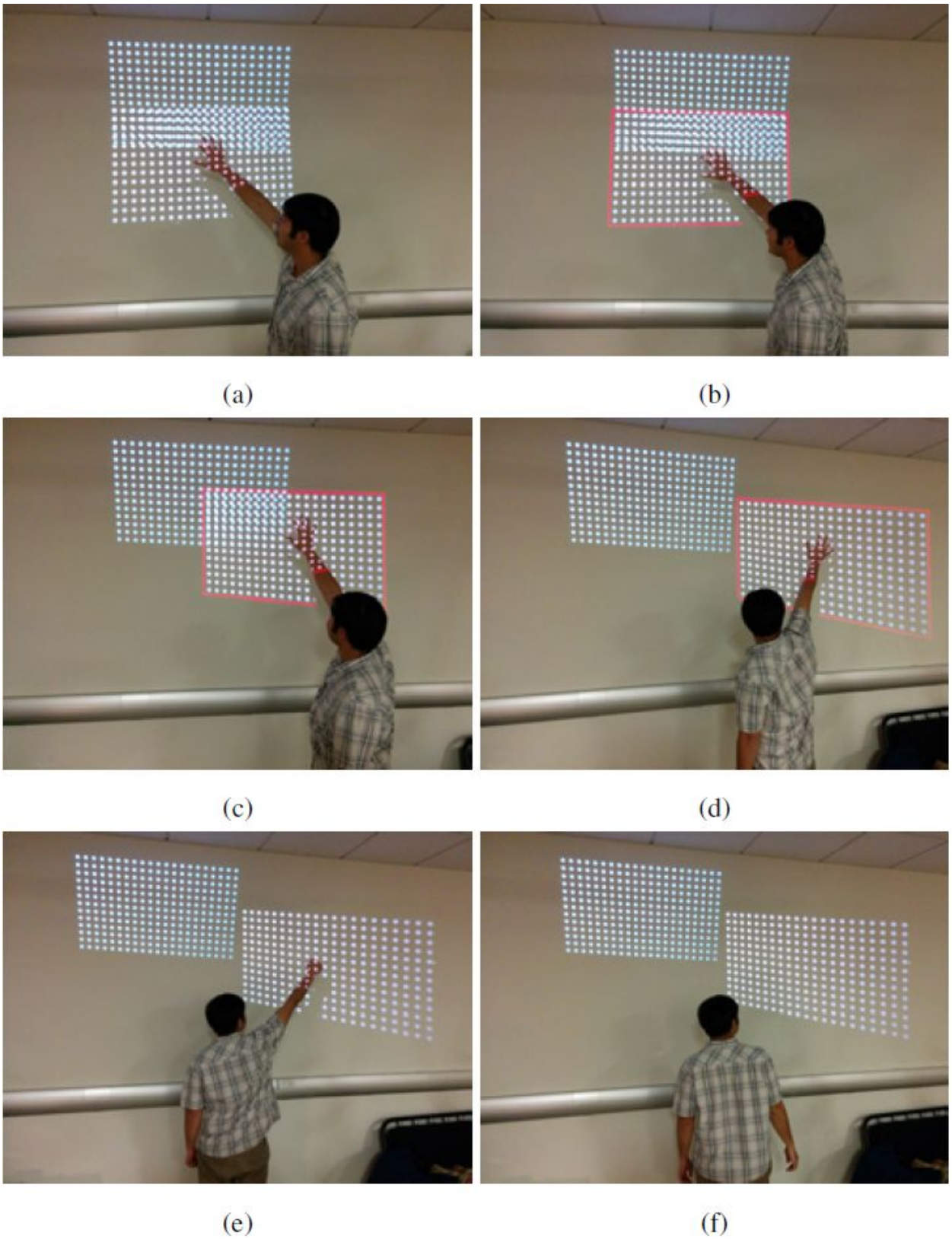


Figure 5.9: (a) User places hand over desired active display he wants to reposition. (b) The active display is selected. (c,d) User moves the active display to a different position. (e) User deselects the active display. (f) The active display is now repositioned.

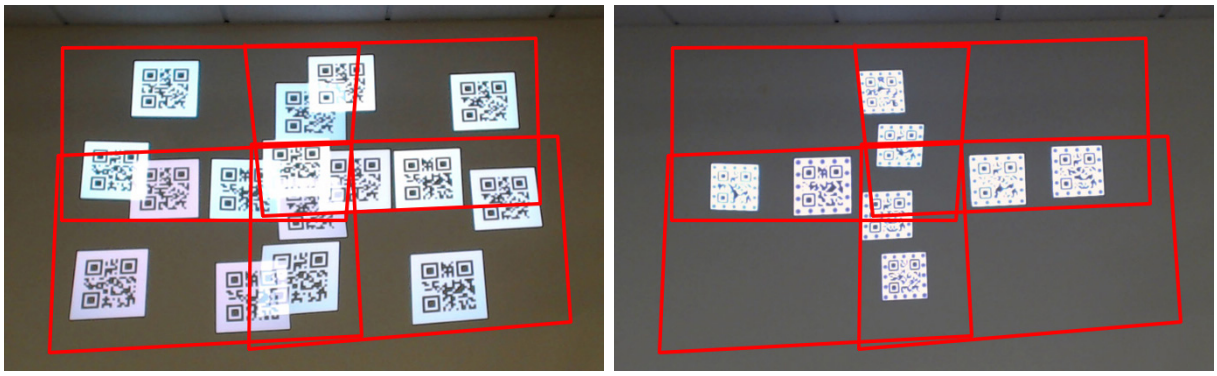


Figure 5.10: The QR codes when placed without reshaping and resizing created conflicts (left). The reshaped and resized QR codes are placed without conflicts (right). The 4 projectors making the CD is shown by the red projector boundaries.

Figure 5.11: This shows a four projector CD before and after registration.

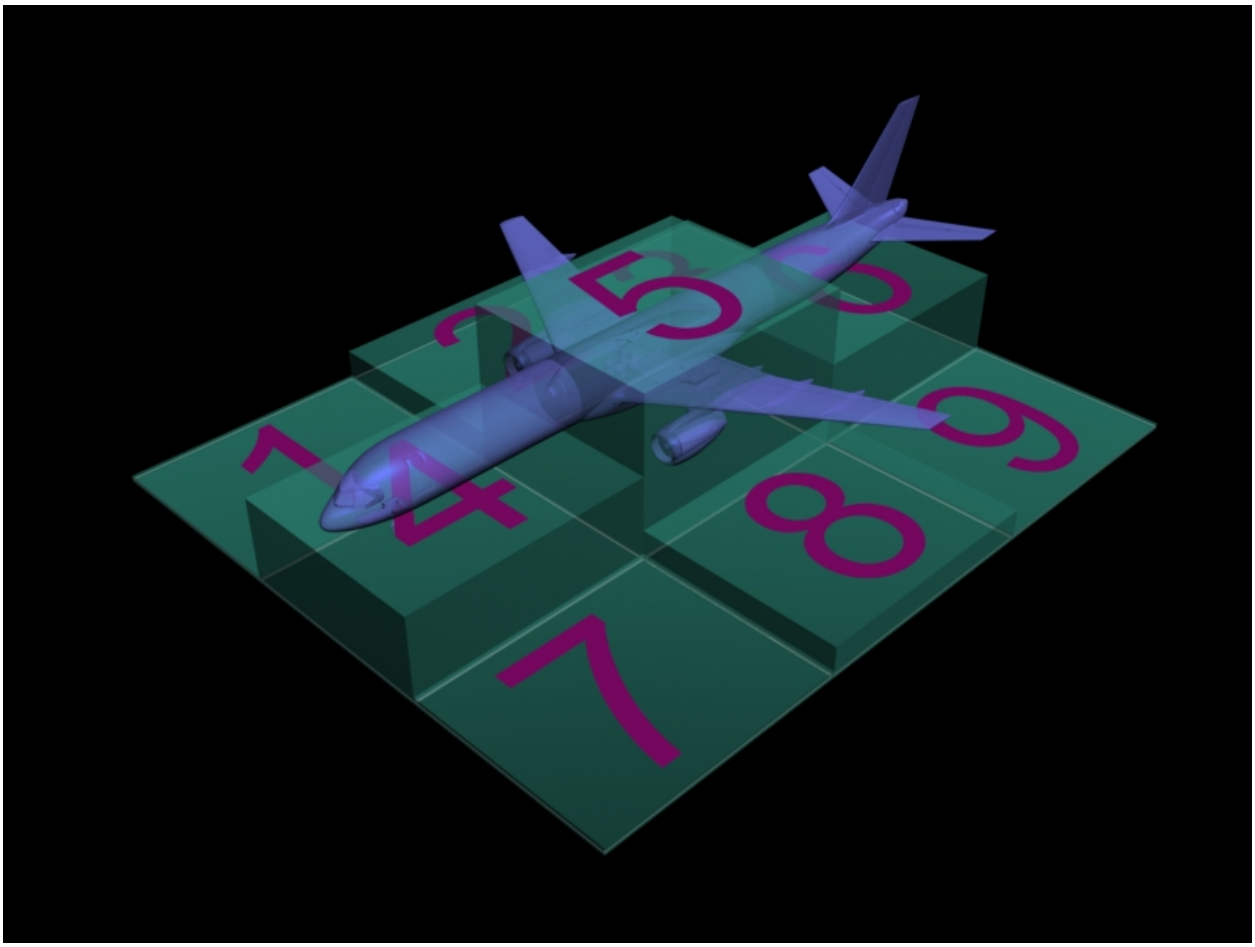
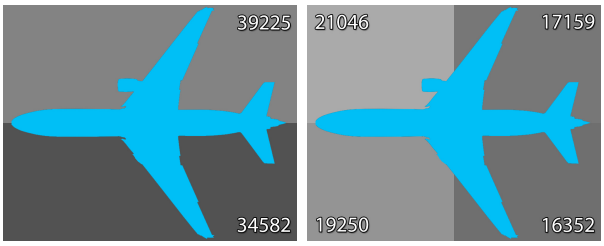


Figure 5.12: 3D data of a Boeing 737 model is partitioned across nine computing units as presented in [29].



(a) (b)

Figure 5.13: *This shows the data load on each machine of a cluster of 2 machines (left) andf 4 machine (right) after interleaved partitioning is used on a 20GB Boeing model. Note that better load balancing is achieved for higher number of machines which has been shown to improve performance in multi-displays in [29].*

Chapter 6

Conclusion

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 - AC), the local external memory (EM) and the external memory of all the other nodes in the distributed system (virtual external memory - VEM). 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.

Building upon the proposed distributed architecture we present a framework of multiple steerable projector-camera unit to create a paradigm of collaborative spatially augmented reality (SAR) workspace. We use the application of modeling of massive 3D data to demonstrate the different collaborative interactions and modalities such a workspace can provide. This opens up the possibility of using such collaborative SAR applications in any workspace shared by multiple users, as is becoming more and more common today.

We presented a specific application of modeling of massive 3D data using our collaborative SAR environment. Note that the front end processes we discussed are responsible for interaction inputs, calibration and display movement. These can be generalized to any other applications especially through the query system that allows users to build a library of gestures to accommodate their needs and application. Further, our front end processes lets application developer query information about the configuration of the system, control of the PTU, and receival of input information. We used the front end process to position the ADN and detect visual input. However, one can use the same processes to achieve other tasks such as tracking and following objects around the room as in the surveillance system, or even as a stereo input device where each ADN looks at an object from different angle and together constructs a 3D model. The back end processes are responsible for the handling and processing of application specific tasks like synchronization and rendering. We used the back end processes towards specific tasks of navigating and interacting with large 3D model. However, one can use the same back end processes to create a display wall or a bulletin board where content from may be arriving from different sources (remote or local) and display them on the CD. Such a system can be used for command and control or other applications where visualization technology for monitoring and decision making is crucial like military, law enforcement and public safety, manufacturing, transportation systems, security and surveillance.

Our work opens up the opportunity to pursue multiple different directions in collaborative workspaces in the future. However, it is important to study how users would adopt and use such workspaces? What kind of interaction modalities are more commonly used than others? What kind of new interaction paradigms can this enable and encourage amidst novel and expert users? This work has been focused on the technology aspects of creating such workspaces. However, we hope that multiple application will be designed using such a workspace to study some of the above mentioned questions in the future.

Bibliography

[1] K. Akeley. Reality engine graphics. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 109–116, New York, NY, USA, 1993. ACM.

[2] W. V. Baxter III, A. Sud, N. K. Govindaraju, and D. Manocha. Gigawalk: Interactive walkthrough of complex environments. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 203–214. Eurographics Association, 2002.

[3] E. S. Bhasker, P. Sinha, and A. Majumder. Asynchronous distributed calibration for scalable and reconfigurable multi-projector displays. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1101–1108, 2006.

[4] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The visual computer*, 17(3):185–197, 2001.

[5] M. Brown, A. Majumder, and R. Yang. Camera-based calibration techniques for seamless multiprojector displays. *IEEE Trans. On Visualization and Computer Graphics*, pages 193–206, 2005.

[6] P. Campbell, K. Devine, J. Flaherty, L. Gervasio, and J. Teresco. Dynamic octree load balancing using space-filling curves. *Williams College Department of Computer Science, Tech. Rep. CS-03-01*, 2003.

[7] X. Cao and R. Balakrishnan. Interacting with dynamically defined information spaces using a handheld projector and a pen. *ACM UIST Symposium on User Interface Software and Technology*, pages 225–234, 2006.

[8] X. Cao, C. Forlines, and R. Balakrishnan. Multi-user interaction using handheld projectors. *ACM symposium on User interface software and technology*, pages 43–52, 2007.

[9] M. Chapman and G. Heiser. vnuma: A virtual shared-memory multiprocessor. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 2–2. USENIX Association, 2009.

[10] P. Chiu. Personal lite display. `http://www.fxpal.com/research-projects/personal-lite-displays/`, 2013. [Online; accessed 2015-02-02].

[11] D. Coetzee, S. Lim, A. Fox, B. Hartmann, and M. A. Hearst. Structuring interactions for large-scale synchronous peer learning. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 1139–1152, New York, NY, USA, 2015. ACM.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2 edition, 2001.

[13] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The cave: Audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, June 1992.

[14] M. F. Deering and S. R. Nelson. Leo: A system for cost effective 3d shaded graphics. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 101–108, New York, NY, USA, 1993. ACM.

[15] J. O. Dorsey, F. X. Sillion, and D. P. Greenberg. Design and simulation of opera lighting and projection effects. In *ACM SIGGRAPH computer graphics*, volume 25, pages 41–50. ACM, 1991.

[16] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):436–452, May/June 2009.

[17] D. Ellsworth. A multicomputer polygon rendering algorithm for interactive applications. In *Proceedings of the 1993 Symposium on Parallel Rendering*, PRS '93, pages 43–48, New York, NY, USA, 1993. ACM.

[18] F. Erol, S. Eilemann, and R. Pajarola. Cross-segment load balancing in parallel rendering. In *Proceedings of the 11th Eurographics Symposium on Parallel Graphics and Visualization*, pages 41–50, 2011.

[19] S. eui Yoon and D. Manocha. Cache-oblivious layouts of bounding volume hierarchies. Technical report, UNC Chapel Hill, 2005.

[20] B. Fitzpatrick et al. Memcached. *Computer Program, available via http://www. memcached. org*, 2010.

[21] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *SIGGRAPH Comput. Graph.*, 23(3):79–88, July 1989.

[22] S. Gallacher, J. O'Connor, J. Bird, Y. Rogers, L. Capra, D. Harrison, and P. Marshall. Mood squeezer: Lightening up the workplace through playful and lightweight interactions. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 891–902, New York, NY, USA, 2015. ACM.

[23] E. Gobbetti and F. Marton. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 878–885. ACM, 2005.

[24] R. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[25] M. Guthe, P. Borodin, and R. Klein. Real-time out-of-core rendering. *International Journal of Image and Graphics*, 2005.

[26] 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. *ACM Transactions on Graphics (TOG)*, 21(3):693–702, 2002.

[27] S. Izadi, H. Brignull, T. Rodden, Y. Rogers, and M. Underwood. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. *ACM UIST Symposium on User Interface Software and Technology*, pages 159–168, 2003.

[28] D.-Q. Lai. Walltop display. `https://www.youtube.com/watch?v=1u09TSfHqZc`, 2015. [Online; accessed 2015-11-10].

[29] D.-Q. Lai, B. Sajadi, S. Jiang, A. Majumder, and M. Gopi. A distributed memory hierarchy and data management for interactive scene navigation and modification on tiled display walls. *IEEE Transactions on Visualization and Computer Graphics*, PP(99), January 2015.

[30] E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the conference on Visualization'99: celebrating ten years*, pages 355–361. IEEE Computer Society Press, 1999.

[31] P. Milgram and H. Colquhoun Jr. A taxonomy of real and virtual world display integration. *ACM SIGGRAPH*, 1999.

[32] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Comput. Graph. Appl.*, 14(4):23–32, July 1994.

[33] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *Computer Graphics and Applications, IEEE*, 14(4):23–32, 1994.

[34] S. Molnar and H. Fuchs. Advanced raster graphics architecture. *Computer Graphics: Principles and Practice*, 1990.

[35] B. Moloney, M. Ament, D. Weiskopf, and T. Moller. Sort-first parallel volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 17(8):1164–1177, 2011.

[36] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: A nonuniform memory access programming model for high-performance computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.

[37] G. Pingali, C. Pinhanez, A. Levas, R. Kjeldsen, M. Podlaseck, H. Chen, and N. Sukaviriya. Steerable interfaces for pervasive computing spaces. *Pervasive Computing and Communications*, pages 315–322, 2003.

[38] C. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. *Ubiquitous Computing*, pages 315–331, 2001.

[39] K. Ponto, T. Wypych, K. Doerr, J. Kimball, and F. Kuester. Videoblaster: A distributed, low-network bandwidth method for multimedia playback on tiled display systems. *IEEE International Symposium on Multimedia*, pages 201–206, December 2009.

[40] R. Raskar, P. Beardsley, J. V. Baar, Y. Wang, P. Dietz, J. Lee, D. Leigh, and T. Willwacher. Rfig lamps: interacting with a self-describing world via photosensing wireless tags and projectors. *ACM Transactions on Graphics*, 23(3):406–415, 2004.

[41] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image based modeling and spatially immersive display. In *Proceedings of ACM Siggraph*, pages 168–176, 1998.

[42] D. Reiners. *OpenSG*. PhD thesis, TU Darmstadt, 2002.

[43] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, et al. Sage: the scalable adaptive graphics environment. *Proceedings of WACE*, 9(23):2004–09, 2004.

[44] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, G. Goldman, J. Leigh, and A. Johnson. Sage: the scalable adaptive graphics environment. *Proceedings of WACE*, 2004.

[45] G. Roman and T. Kimura. A vlsi architecture for image composition. *Computer Graphics: Principles and Practice*, pages 113–118, 1979.

[46] P. Roman, M. Lazarov, and A. Majumder. A scalable distributed paradigm for multi-user interaction with tiled rear projection display walls. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1623–1632, 2010.

[47] M. Roth, P. Riess, and D. Reiners. Load balancing on cluster-based multi projector display systems. In *14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 55–62, 2006.

[48] B. Sajadi, Y. Huang, P. Diaz-Gutierrez, S. Yoon, and M. Gopi. A novel page-based data structure for interactive walkthroughs. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 23–29, New York, NY, USA, 2009. ACM.

[49] B. Sajadi, Y. Huang, P. Diaz-Gutierrez, S.-E. Yoon, and M. Gopi. A novel page-based data structure for interactive walkthroughs. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 23–29, 2009.

[50] B. Sajadi, S. Jiang, M. Gopi, J.-P. Heo, and S.-E. Yoon. Data management for ssds for large-scale interactive graphics applications. In *Symposium on Interactive 3D Graphics and Games*, pages 175–182. ACM, 2011.

[51] B. Sajadi, M. Lazarov, M. Gopi, and A. Majumder. Color seamlessness in multi-projector displays using constrained gamut morphing. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1317–1326, 2009.

[52] B. Sajadi and A. Majumder. Markerless view-independent registration of multiple distorted projectors on extruded surfaces using an uncalibrated camera. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1307–1316, 2009.

[53] B. Sajadi and A. Majumder. Auto-calibration of cylindrical multi-projector systems. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 155–162. IEEE, 2010.

[54] B. Shneiderman. Response time and display rate in human performance with computers. *Computing Surveys*, 16(3):265–285, 1984.

[55] A. Simon. First-person experience and usability of co-located interaction in a projection-based virtual environment. *ACM Symposium on Virtual Reality Software and Technology*, pages 23–30, 2005.

[56] T. Sueishi, H. Oku, and M. Ishikawa. Robust high-speed tracking against illumination changes for dynamic projection mapping. In *Virtual Reality (VR), 2015 IEEE*, pages 97–104, March 2015.

[57] M. Tchiboukdjian, V. Danjean, and B. Raffin. Binary mesh partitioning for cache-efficient visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(5):815–828, 2010.

[58] J. Underkoffler. A view from the luminous room. *Personal Technologies*, 1(2):49–59, 1997.

[59] C. Wang, J. Gao, and H. Shen. Parallel multiresolution volume rendering of large data sets with error-guided load balancing. In *Proceedings of the Eurographics Parallel Graphics and Visualization Symposium*, pages 23–30, 2004.

[60] S. Whitman. A task adaptive parallel graphics renderer. In *Proceedings of the 1993 Symposium on Parallel Rendering*, PRS '93, pages 27–34, New York, NY, USA, 1993. ACM.

[61] S.-E. Yoon, B. Salomon, R. Gayle, and D. Manocha. Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *Visualization and Computer Graphics, IEEE Transactions on*, 11(4):369–382, 2005.