A Scalable Distributed Paradigm for Multi-User Interaction with Tiled Rear Projection Display Walls

Pablo Roman and Maxim Lazarov and Aditi Majumder, *Member, IEEE*

Fig. 1. This figure shows some of our applications in action. From left to right: Our collaborative map visualization application with two users visualizing different parts of the map at the same time on our 3×3 array of nine projectors; Our collaborative emergency management application with two users trying to draw a path to hazardous location and dispatching teams of first responders on our 3×3 array of nine projectors; Digital graffiti drawn using our collaborative graffiti application on only six of the projectors. We deliberately did not edge blend the projectors to show the six projectors clearly; Four children working together on our digital graffiti application on a 3×3 array of nine projectors.

Abstract— We present the first distributed paradigm for multiple users to interact simultaneously with large tiled rear projection display walls. Unlike earlier works, our paradigm allows *easy scalability across different applications, interaction modalities, displays and users*. The novelty of the design lies in its distributed nature allowing well-compartmented, application independent, and application specific modules. This enables adapting to different 2D applications and interaction modalities easily by changing a few application specific modules. We demonstrate four challenging 2D applications on a nine projector display to demonstrate the application scalability of our method: map visualization, virtual graffiti, virtual bulletin board and an emergency management system. We demonstrate the scalability of our method to multiple interaction modalities by showing both gesture-based and laser-based user interfaces.
Finally, we improve earlier distributed methods to register multiple projectors. Previous works need multiple patterns to identify the neighbors, the configuration of the display and the registration across multiple projectors in logarithmic time with respect to the number of projectors in the display. We propose a new approach that achieves this using a single pattern based on specially augmented *QR codes* in constant time. Further, previous distributed registration algorithms are prone to large misregistrations. We propose a novel *radially cascading geometric registration technique* that yields significantly better accuracy. Thus, our improvements allow a significantly more efficient and accurate technique for distributed self-registration of multi-projector display walls.

Index Terms—Tiled Displays, Human-Computer Interaction, Gesture-Based Interaction, Multi-user interaction, Distributed algorithms

1 Introduction

Large multi-projector planar display walls are common in many visualization applications. We have seen a large amount of work on camera-based registration of multiple projectors in such displays, both for geometry and color [25, 8, 4, 21, 22, 2, 3, 28]. This has enabled easy deployment and maintenance of such displays. However, a suitable interaction paradigm for these displays that can be scaled to multiple users for large number of display modules across different applications and interaction modalities is still not available. This has brought in a belief in the visualization community that limited display space and interactivity makes it difficult for application users to solve issues of interactively understanding domain problems. This paper focuses on providing a new approach of scalable interactive multi-user interaction for tiled display walls. The final roadblock in the adoption of any technology is the ease with which users can interact with it. Our scalable interaction paradigm brings in the hitherto unknown ease in user interactivity and deployment of it on commodity projectors. Hence, this work can have a significant impact on wider adoption of the seamless multi-projector display technology across the visualization community.

Most work in the human computer interaction domain [6, 31, 30, 32, 36, 29, 20, 19, 17, 16, 37, 9, 7] is difficult to scale to multiple interaction modalities, applications, users and displays. Central to this problem is the fact that almost all earlier works in the domain of interaction with tiled displays have explored application specific centralized algorithms and architectures which inherently cannot scale with respect to the number of users and displays due to critical dependency on a single server. Further, scalability to multiple applications and interaction modalities demand careful algorithm design to compartmentalize the application/interface specific modules from the application/interface independent ones and has not been explored before.

This paper makes the first effort to design a scalable interaction paradigm for rear-projected tiled displays that can scale with multiple projectors, users, applications and even interaction modalities. We observe that such a general paradigm is only possible with a distributed architecture that inherently provides mechanisms for scalability. Such a distributed architecture for multi-projector display walls is presented in [3] where the display is built by a distributed network of plug-and-play projectors (PPPs). Each PPP consists of a projector, a camera and a communication and computation unit, simulated by a computer. The display is created by a rectangular array of these PPPs on a planar screen. Each PPP runs an SPMD (single program multiple data) algorithm presented in [3] that starts by believing that it is the only display in the environment. It can communicate with its neighbor using its camera which sees parts of its neighboring PPPs. Using such visual communication via the cameras and a distributed configuration

• *Pablo Roman and Maxim Lazarov and Aditi Majumder are in Computer Science Department of University of California, Irvine, E-mail: {proman,mlazarov,majumder}@uci.edu.*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

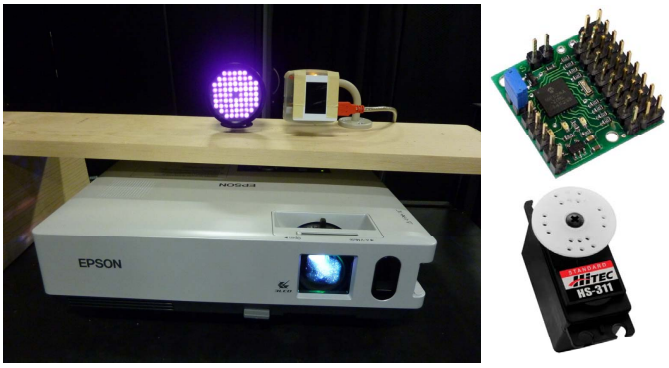

Fig. 2. Left: Our prototype PPP with a projector, a camera and a computer; Right bottom: The inexpensive RC servo that can be used to switch the IR filters back and forth. Right top: The RS-232 8 servo controller.

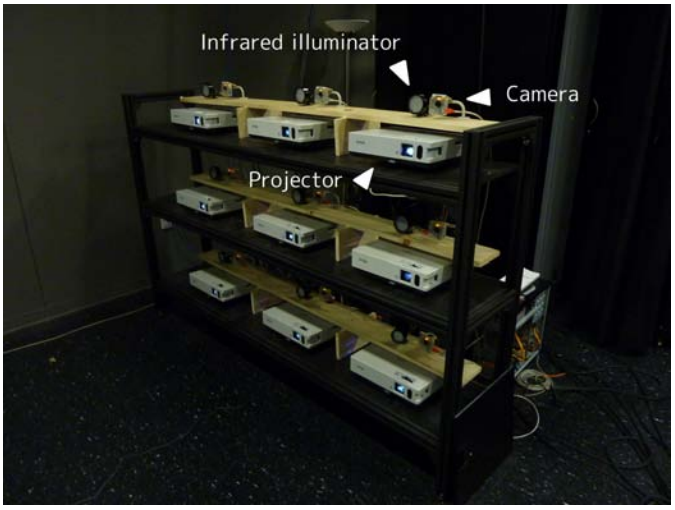

Fig. 3. Our setup of a network of PPPs augmented by the IR illuminators and the IR camera filters.

discovery algorithm, the PPPs discover the total number of PPPs creating the tiled display, their array configuration (total number of rows and columns) and its own coordinates (its own row and column) in this array. Following this, the PPPs can align themselves to create a seamless display using a distributed self-registration algorithm.

1.1 Main Contributions

We use the same distributed architecture based on PPPs presented in [3] and build a new distributed registration algorithm and a distributed interaction paradigm on top of it. For interaction, we design a *SPMD distributed interaction algorithm* that runs on each PPP following the registration to allow multiple users to interact with the display using any kind of interaction modality. The highlights of our distributed interaction algorithm are as follows.

1. Since we design an SPMD algorithm, it can easily *scale to multiple projectors*. Hence, adding and removing PPPs to reconfigure the display does not necessitate any change in the interaction algorithm.

2. Most modules of our algorithm are application independent. Hence, to adapt to different 2D applications, only a few application specific modules need to be modified. This allows our algorithm to *scale to many 2D applications*.

3. Similarly, changing the interaction modality requires modifying a small number of interface dependent modules. This allows our algorithm to *scale to different interaction modalities* as well (e.g. laser pointers, gesture-based interface).

4. Unlike a centralized system where all the interaction from multiple users is handled by a single centralized server, a distributed algorithm distributes the load of handling multiple users to multiple PPPs. Hence, our algorithm can easily *scale to multiple users*.

We also propose a new distributed registration technique that achieves much greater accuracy and is more efficient in terms of performance and bandwidth load than the method presented in [3]. Below are the highlights of our new registration technique as compared to [3].

1. First, while discovering the configuration of the PPP array, multiple rounds of visual communication were used via the cameras in [3]. This required processing multiple patterns for each PPP and converged in logarithmic time with respect to the number of projectors in the display. The performance was also compromised due to computationally intensive image processing. In contrast, we design a novel method in which each PPP uses a single pattern made of specially augmented *QR (Quick Response) codes* to discover the display configuration and self-register simultaneously in constant time. More importantly, we achieve this without increasing the network communication load across the PPPs.

2. Second, [3] uses a distributed homography tree algorithm for self-registration of the PPPs. This can lead to large misregistrations (even as large as 10-20 pixels), especially when the PPPs are further away from the reference PPP. This impacts the scalability of the self-registration algorithm to a large number of projectors. We present a novel *radially cascading geometric registration* method that can achieve a much superior accuracy.

In summary, our work, for the first time, *introduces* an entirely *distributed framework for user interaction* with tiled displays. In addition, we *improve* the existing *distributed framework for registering* the many PPPs in the display. We first discuss our system in detail in Section 2, followed by the distributed interaction paradigm and the improved distributed registration in Section 3 and Section 4 respectively, concluding with future directions in Section 5.

2 System Overview

Our system consists of N PPPs, each made of a projector, and a camera connected to a computer. We assume that the projectors and cameras are linear devices. The PPPs are arranged casually in a rectangular array (Figure 3) and overlap with their neighbors (adjacent PPPs). The PPPs are initially unaware of the configuration of the array that they are arranged in. Using visual communication via the cameras, a PPP starts detecting its neighbors whenever its associated camera perceives some other PPP in its overlapping coverage area with the adjacent PPP. Using our distributed registration technique (Section 4) each PPP can discover its neighbor, the total number of projectors in the display and their array configuration, its own coordinates in the array of PPPs and finally self-register itself with other PPPs to create a seamless display. The PPPs use an NTP (Network Time Protocol) clock synchronization to achieve a synchronized clock across the multiple PPPs. The importance of such a synchronization will be evident in the subsequent sections when we describe our distributed interaction paradigm. We also assume that the PPPs use a constant IP multicast group to communicate.

Once the PPPs are registered, we desire to interact with the display. We use two kinds of interaction modalities in this paper – 2D hand gestures and laser based interaction. Though the lasers are bright enough to be detected easily in the projected images, when using gestures the camera cannot detect visible light gestures reliably because the screen and projected image obstruct the hand. To handle this situation, as in [12, 19], we augment our PPP with an IR illuminator and mount an IR bandpass filter on the camera. These filters are removed during registration and then put back to resume interaction. The IR illuminator and the IR filter on the camera allow us to detect gestures when we touch the screen. We use a standard rear-projection screen (from Jenmar), which acts as a good diffuser of IR light. In our setup, we use monochrome cameras without IR cut filters, although we only used one of the color channels. Figure 2 shows one of our IR sensitive PPPs and Figure 3 shows our setup. Removing the IR filter during registration can be achieved automatically by inexpensive RC servos

($10/unit) and can be controlled with serial (RS-232) servo controllers ($20 for controlling 8 RC servos), which are also simple and inexpensive. The IR emitter must also be switched off during registration which could be done via a serial/USB-actuated relay. This can prevent the sensor from getting saturated by both IR and projected visible light.

3 The Distributed Interaction Paradigm

In this section, we describe our distributed interaction paradigm in detail. We start by describing the related work in the domain of various interaction paradigms for large scale displays and comparing our work with it (Section 3.1). Next we describe our distributed algorithm in detail in Section 3.2. When describing this, we consider 2D gesture-based interaction since restricting to a specific interaction modality allows us to provide a simple explanation. However, we explain ways to scale to different interaction modalities in the end of Section 3.2.1. We present implementation details and results in Section 3.3 and 3.4.

3.1 Related Work

Large displays require interaction modalities that match their scale instead of the more traditional mouse and keyboard. The most natural form of such an interaction is using gestures and several works have explored it [1, 11, 29]. Since detecting a gesture unambiguously is a difficult computer vision problem, touch sensitive surfaces [19, 27, 37, 10, 12] have been explored for better localization of gesture dependent features. New devices that can aid when gestures are ambiguous have also been explored [34]. Parallely, we have seen the development of interaction devices by which users can convey their intentions much more precisely without the ambiguity of gestures. These include devices like simple lasers [24], VisionWand [5], a special touch pad [23], LEDs on tangible devices [20], a remote control [18], objects with simple geometry like blocks or cylinders [33], or even a handheld camera [17]. However, all these works focus on interfaces and hence use a simple single display and single sensor paradigm.

When using multiple projectors and sensors, new issues arise like tracking the interaction across multiple projectors and cameras, deciding on a reaction that is unanimous across the multiple projectors, and reacting in minimal time using minimal network bandwidth. There have been a few works that use multiple projectors, but they use a single camera or a pair of stereo cameras. Hence, the interaction tracking in these systems is centralized and handled by a single server [18, 35, 36, 20]. Further, the same centralized server decides a suitable reaction to the gesture and informs the different projectors on how to react. Though this does not mean that all projectors react similarly, a centralized server decides and communicates the different reaction each projector should produce.

Few recent works address systems with multiple projectors and cameras. [9] uses a laser based interaction paradigm where multiple cameras can detect the location of multiple lasers used by multiple users. [30] uses multiple cameras to detect gestures of multiple people. Although in both these systems the processing of images from the camera is done in a distributed manner by a computer connected to each camera, the processed data is then handed to a server that finds the 2D position of the gesture directly or by triangulation. The same server is responsible for managing the projectors and hence it decides the reaction for each projector and communicates it to them. Thus, these works all have in common the centralized architecture where a single server is responsible for tracking the gesture and then reacting to it. Distributed rendering architectures [13, 14, 15] also follow a similarly centralized architecture where the rendering takes place in a distributed manner in computers attached to each projector, but they are controlled by a centralized server that manages how the rendering should be distributed.

Comparison with Our Work: In contrast, our work focuses on a completely distributed paradigm where each PPP acts as self-sufficient module. Unlike previous work, where the projectors and cameras are treated as different devices, we view the display as a distributed network of PPPs. Our goal is to develop a single program multiple data (SPMD) algorithm to be run on each PPP that would detect and track the user action in a completely distributed manner affecting only the PPPs that see the action. Further, an appropriate reaction should be produced by the relevant PPPs in response to the gesture, even if the gesture does not occur within them. This assures minimal network bandwidth usage since all PPPs do not communicate to a single centralized server and minimal time since the processing is shared by multiple PPPs and is not the responsibility of a single centralized server. Such a distributed paradigm allows easy scalability to multiple displays, applications, interaction modalities and users.

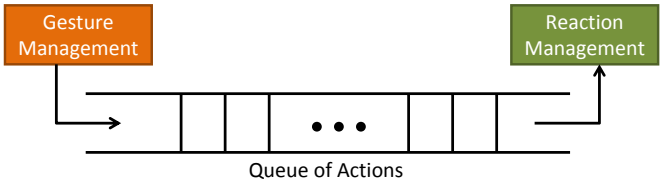

Fig. 4. The Distributed Interaction Paradigm.

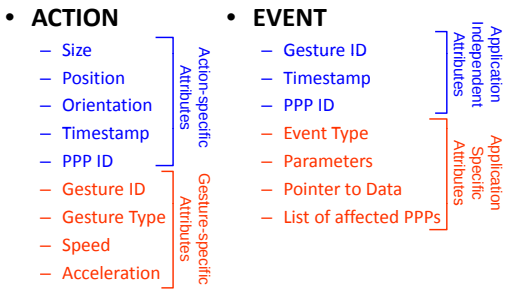

Fig. 5. This figure describes the action data type used in gesture manager and the event data type used in reaction manager.

3.2 The Algorithm

We consider interaction to be a set of two operations that occur consecutively: (a) a *2D gesture* made by the user; and (b) a *consequent reaction* provided by the display. We assume that a gesture is a sequence of samples, also called *actions*, detected by the system. These samples can be generated through a multitude of input systems including touch – by placing the palm on the screen, or laser pointers. The meanings of isolated or temporal sequences of actions are predefined by applications for consistent interpretation. Note that since a gesture occurs over an extended period of time, it can span across multiple PPPs moving between the non-overlapping and overlapping areas of the PPPs. Further, it is important that the reaction does not wait for the gesture to complete. For example, if the user is moving his/her hands from left to right, he/she is expecting the underlying image to move from left to right even before he/she completes the gesture. Hence, the goal is to identify the gesture even when it is not complete and start reacting as soon as possible.

Our distributed interaction paradigm consists of two main components: a *distributed gesture management* module (Section 3.2.1) and a *distributed reaction management* module (Section 3.2.2). These are run as two threads in a producer-consumer fashion in each PPP (Figure 4). The distributed gesture management module produces a queue of actions that are then processed (or consumed) by the distributed reaction manager in an FCFS manner. Note that though the user's intentions are interpreted per gesture (which is a series of actions), the processing of these gestures is done per action. This difference in the granularity of interpretation and processing allows the system to respond to a gesture as soon as it commences without waiting for its end. Finally, the distributed gesture management is application independent. The application specific modules occur only during reaction management.

Fig. 6. This figure shows a few different types of hand postures used for gesture-based interaction. Each application can define its own interpretation for each posture. For example, in our map visualization application, touching the screen with two fingers is used to change the displayed layer, touching it with one finger is used to open and close individual working windows, sweeping with an open hand is used to move the map around and twisting a closed hand is used for zooming in and out. On the other hand, in our graffiti application, two fingers are used to bring up a color palette, one finger to select a color from the palette and any other postures to draw lines.

3.2.1 Distributed Gesture Management

In a distributed network of PPPs, there is no centralized server that manages the observed actions of the user. Each PPP is responsible for managing the actions that occur within its domain. When the gesture spans across multiple PPPs, we design a mechanism to track it and hand over its management from one PPP to another as it moves across PPPs. This is achieved in a manner transparent to the user. Further, our framework can handle *multiple* users each doing single hand gestures.

The distributed gesture management involves (a) a *shared action management mechanism* to decide which PPP handles which part of the gesture via their reaction managers, and (b) *shared gesture tracking* to follow the path of the gesture as it moves across multiple PPPs and is facilitated via an anticipatory action communication mechanism.

Action Data-Type: First we describe the action data structure (Figure 5) filled up by a PPP when it detects an action. This consists of *action specific attributes* like position, orientation and size of the hand, detecting PPP ID and timestamp (synchronized by NTP) in the detecting PPP. The timestamp needs to be included in the attributes to handle race conditions, described in detail in Section 3.2.2. The position is denoted in the global coordinates of the display. Note that since each PPP knows the exact configuration of the entire display and its position in it in the post-registration phase, it can easily calculate the global position of the action. The action also contains some *gesture specific attributes* like gesture ID, gesture type, speed, and acceleration. As soon as the commencement of a new gesture is identified, a new gesture ID is generated. When detecting the ith gesture in the jth PPP, $0 \leq j < N$, the PPP assigns a gesture ID of $i*N+j$. Hence, the identity of the PPP where the gesture commenced can be deciphered from the gesture ID field of the first action in that gesture. Gesture type refers to the type of hand posture which when seen over a period of time constitutes the gesture (Figure 6). The speed and acceleration of the gesture denote its local speed and acceleration at the time when this component action was made in the gesture. The speed and acceleration is computed by finding the differences of the position and speed respectively in two consecutive actions in a gesture.

To detect a gesture, the PPP first recognizes the first action of the gesture in its camera frame. At the commencement of the gesture, the gesture type is set to be `undecided`. To detect the gesture type robustly and reliably, a few of the initial actions are examined. Each of these actions votes for one particular gesture type. The gesture type that receives the maximum votes is selected as the type of the gesture.

Shared Action Management: The shared action management scheme enqueues the constituting actions of a gesture as it moves across the display through multiple PPPs. When an action that does not belong to any previous gesture is detected, it indicates the commencement of a *new gesture*. If this new action or part of it occurs in the non-overlapping region of a PPP, since no other PPP can see this action completely, this PPP bears the responsibility to enqueue this action to be processed by its reaction manager. However, when the action occurs in the overlap region of multiple PPPs, it is desirable for only one PPP to enqueue it for processing by the reaction manager. This avoids inconsistent reaction from multiple PPPs. To assure this, when in the overlap region of multiple PPPs, the gesture is only handled by the PPP with the smallest ID. Figure 7 illustrates this process.

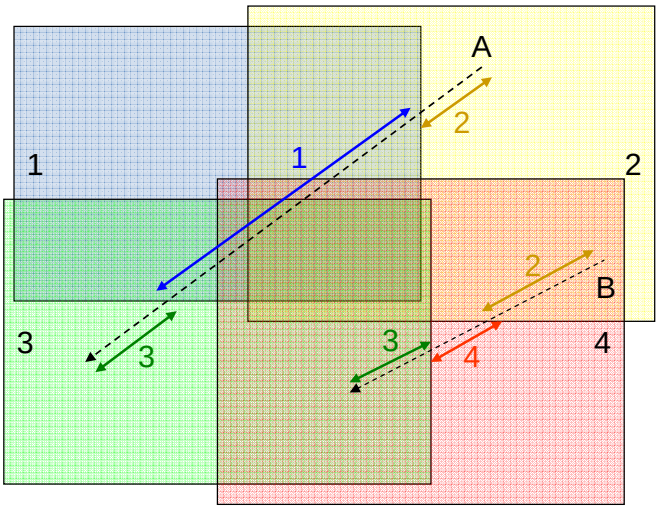

Fig. 7. This figure shows how the gestures made across multiple PPPs are handled in a shared manner by multiple PPPs in their lifespan. A and B denotes two different gestures. The length of the gesture is divided and labeled to show which PPPs handle which part of the gesture.

A and B denote two gestures. A starts in the non-overlapping area of projector 2. As soon as it enters the overlapping region of 1 and 2, 1 picks up the gesture since it has the smaller ID of the two projectors. After this, note that 1 continues to handle the gesture even though it moves through the overlap of 1 and 2, overlap of 1 and 4, overlap of all four projectors, overlap of 1, 3 and 4, and overlap of 1 and 3. Only when the gesture moves to non-overlapping area of 3, it is handled by 3 since no one else can see it. Similarly, in gesture B, when it starts in the overlap of 2 and 3, it is first picked up by 2. Then it is handled by 4 in the non-overlapping area of 4. But as soon as it moves to the overlapping area of 4 and 3, 3 starts to handle the gesture.

Shared Gesture Tracking: The gestures are tracked in a shared manner by multiple PPPs when they span multiple PPPs. This is achieved via the temporal and spatial proximity of consecutive actions in a gesture. If an action is close temporally and spatially to another action, it is assigned the same gesture ID and type. If an action is temporally or spatially far away, it is considered the commencement of a new gesture. This can happen when two users are interacting simultaneously with the same PPP. For this purpose, a threshold has been defined that tries to make a compromise between allowing fast gestures and correctly separating different gestures.

When the gesture management migrates from one PPP to another, we use an *anticipatory action handover* mechanism to handle it. When a PPP is tracking the gesture within itself and finds it is to move into the neighborhood of an adjacent PPP, it sends an anticipatory message to notify the neighboring PPP about a gesture coming its way. This message contains all the action data necessary to handle the continuation of a gesture: position, PPP ID, gesture ID, gesture type, speed, acceleration and timestamp. Using the position, speed and timestamp, the receiving PPP can match it against future detected actions by assuming that the gesture continues at a similar speed and acceleration. In a later instant in time, when the adjacent PPP detects an action in the neighborhood of the location predicted by an anticipatory action message, it identifies the action as part of a *continuing gesture* and hence copies its gesture-specific attributes from this anticipation message. Following this, the new PPP starts tracking and managing the actions of this gesture. However, note that between a prediction and actual detection of the action in the adjacent PPP, multiple actions can occur. Hence, the adjacent PPP receives multiple anticipation messages from the same neighbor. When processing them, it only needs to consider the most recent anticipatory message. Also, if a PPP receives anticipation messages from multiple PPPs due to multiple gestures, they can be easily identified by their PPP ID attribute. The end of a gesture is

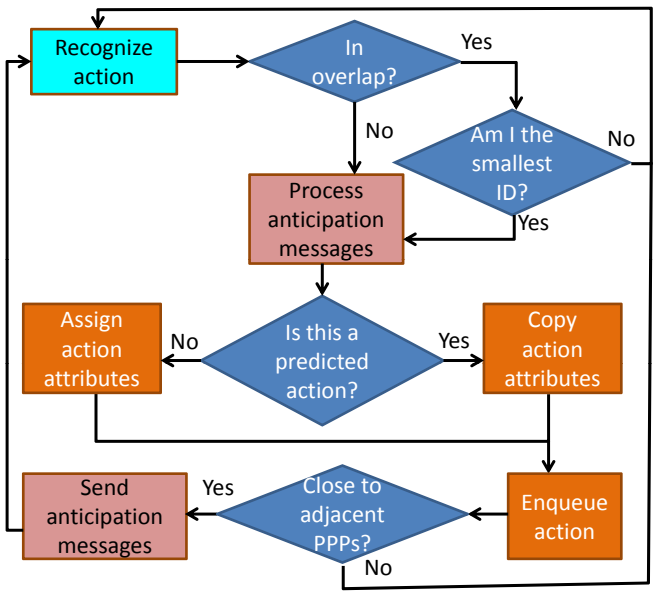

Fig. 8. The Distributed Gesture Management Protocol: The gesture management depends on the 2D application only on interpretation of recognized gestures and hence is mostly an application independent module. The cyan boxes represent the application specific modules and the purple boxes represents modules that are involved in communications.

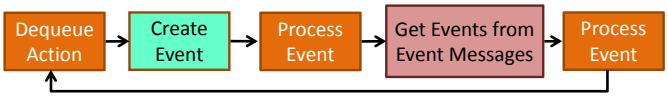

Fig. 9. The Distributed Reaction Management Protocol. The cyan boxes represent the application specific modules and the purple boxes represents modules that are involved in communications.

detected by a timeout mechanism. If the difference in timestamp of two consecutive actions is beyond a certain threshold, the new action is assumed to be the commencement of a new gesture.

Flow Chart: The entire gesture management module is summarized in the flowchart in Figure 8. Each PPP starts with detecting an action and deciding to pick up its management using the shared action management protocol. If the gesture continues within itself, the PPP tracks it. If the gesture moves close to an adjacent PPP, it communicates it to the relevant neighbor via the anticipatory action message. And if it receives an anticipatory action message, it picks up the gesture tracking and handling from an adjacent PPP.

Scaling to Different Interaction Modalities: To use different interaction modalities, only the cyan box in Figure 8 that recognizes the actions need to change. Instead of gesture based action, this module has to now detect different kinds of actions.

3.2.2 Distributed Reaction Management

The distributed reaction mechanism involves processing (consuming) the actions in the queue generated by the distributed gesture manager by reacting with a particular event. Note that the set of PPPs that need to react to a gesture may be larger than the set of PPPs across which the gesture spans. For example, in a map visualization application one can move the map with a sweeping gesture that spans just a few PPPs, but the map across all PPPs must move in response. Further, the event may be associated with creation, movement, or deletion of data across PPPs. Hence, the reaction manager is also responsible for taking steps to assure data consistency across the PPPs. Finally, the job of the event manager also involves informing the PPPs that will be affected by the event so that reaction managers of the PPPs that did not see any gestures can receive events they need to perform from

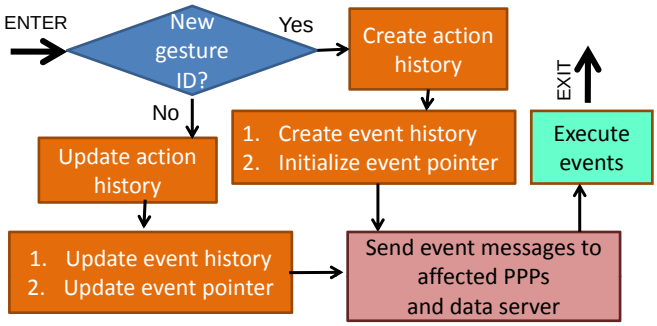

Fig. 10. Details of process events routine in the Reaction Manager in Figure 9. The cyan boxes represent the application specific modules and the purple boxes represents modules that are involved in communications.

other PPPs. The function of the reaction manager is summarized as in Figure 9. It dequeues an action from the queue of actions produced by the gesture manager and creates and processes the corresponding event. Following that, it checks if it has received any events from other PPPs and processes them. The details of the events processing is described later in this section (Figure 10).

Event Data Type: Figure 5 shows the event data structure used by the reaction manager. Processing every action invokes an event. Hence, just as every gesture in the gesture manager is associated with a set of actions, every gesture in the reaction manager is associated with a set of events. The event attributes constitute of a gesture ID, a timestamp indicating when the event was invoked, and the PPP ID of where it was invoked. These are all application independent attributes of an event. The application specific attributes of the event are its type and parameters. For example, the event type may be something as simple as move or rotate or as complex as open a window or resize a window. The event parameters can be the horizontal or vertical dimensions for a move, an angle for rotation and so on. Event parameters also contain a pointer to data which the event has created or is handling. For example, for opening a window, the data is the window (that can be defined by a top-left and bottom-right corner in global coordinates of the display). Finally, the event also maintains a list of PPPs it affects (e.g. for a global event like panning the imagery across the display, this will be the set of all PPPs in the display).

Event and Action History: As soon as the reaction manager gets an action from the queue, it creates an event. The reaction manager deciphers this predefined application specific event type associated with the action. The reaction manager can also receive events which it has to execute from other PPPs. For processing the events, the reaction manager maintains an *event history* and *event pointer* for every gesture it encounters. The event history is the array of all the events invoked by the actions in a gesture, sorted by their creation timestamp. The *event pointer* marks an index in the event history after which the events have still not been executed. This event history is instrumental in performing the events associated with a gesture in the same order in all the affected PPPs. As detailed later, the PPP may sometimes encounter an event out-of-order after it has executed some of the subsequent events. In such scenarios, the event history will be used to trigger an event 'roll back' procedure. This action history can be as large as the array of all actions comprising a gesture or as small as the previous action in the same gesture. The size of the action history depends entirely on the events that the application needs to perform. Its purpose is to enable the generation of different events depending on a sequence of actions, like detecting hand-drawn letters.

Event Processing: In order to process the event (Figure 10), the reaction manager first checks if the action belongs to a gesture that the PPP has not seen before. This indicates a new gesture and the reaction manager creates a new action history, an event history (each of them containing one of the two data types in Figure 5) and initializes the event pointer to the beginning of the event history. Next, the reaction manager computes the event attributes of this new event associated

with this gesture. Note that for some events, like opening a window by drawing a stroke from the top left to the bottom right corner of the window, it may not be possible to find the list of all the affected PPPs with the first action of the gesture. In this case, the PPPs will update the data attributes in a shared manner and inform the relevant PPPs as the gesture proceeds across the different PPPs. This would also mean updating the event history, action history and the event pointer in an appropriate manner. Finally, if some new data has been written, the PPP will also commit this change to the data server so that when other PPPs request the part of the data at a future time, they can see the updated data. Note that since multiple PPPs are accessing and writing data from and to a data server we assume all the different mechanisms are in place to assure consistency in a distributed data management application. Hence, our interaction paradigm is agnostic to the kind of data server being used, either centralized or distributed. Following this the reaction manager proceeds to execute the event. Executing the event involves performing the function associated with the event (moving the imagery or showing a new window and so on) and advancing the event pointer in the event history. Then it sends a message with the event data structure to all the PPPs currently in the list of affected PPPs that should perform a consistent event (e.g. moving the image by an equal amount).

Race Conditions: Finally, there may be a situation when an action is being processed by the reaction manager while an event related to a prior action in the same gesture is waiting in the message buffer. Hence, when the PPP gets to process the event message, it arrives out-of-order with respect to the other events in the event history for the particular gesture. Now, for certain events this may not be important since the attributes may be changed in such a manner by the subsequent events that it is not inconsistent with respect to this PPP. But, if this reveals a data inconsistency, then we need to execute an event 'roll back' procedure. In the 'roll back' procedure, we undo the effects of all the events in the event history that have a larger timestamp than the timestamp of the received event and reverse the event pointer appropriately to reflect this. Then the received event is inserted in the event history at the position of the reversed event pointer and executed. Finally, all the subsequent events are executed again and the event pointer is advanced accordingly. If there is more than one gesture affecting the same data in a manner that can cause data inconsistency, all the events with a bigger timestamp in the multiple gestures will have to be rolled back in the same manner – undone in a reverse timestamp order and executed again in the timestamp order. Old events can be removed from the event history when newer packets have been received from all the PPPs (TCP based communication ensure delivery in order) or when the event is older than the connection timeout time. Though the case of out-of-order event does not occur very often, this 'roll back' operation is critical to ensures that the final logical order of event execution is consistent across the PPPs and hence the data. One example of the occurrence of this is when a gesture is right on the division between PPPs and small registration errors result in both PPPs handling the gesture. In this case, the gesture will be treated as rapidly going from one PPP to another. The messages will be received out of order but will be correctly reordered by the 'roll back' procedure. Since our registration is very accurate, this does not produce perceivable effects in the application. Since this procedure modifies actual data, its effects can be sometimes perceived by the users. For example, in the graffiti application described below and shown in the accompanying video, when crossing the border between PPPs a line will sometimes be drawn for a split second in the wrong order (as going back and forth between PPPs) before quickly correcting itself. This will, however, happen rarely and be quick enough to not be a nuisance.

Scaling to Different Applications: Note that only a few modules of the reaction manager are application specific (highlighted in cyan boxes in flowcharts in Figure 9 and Figure 10). The design of the event attributes and types depends on the application and hence so does their assignment during event creation. Further, the way events are executed is also application specific. The rest of the reaction manager is common for all kinds of 2D applications and is hence application independent. The application specific modules for our test applications are explained in detail in Section 3.4.

Table 1. This table represents the network usage in amount of packets and bytes per second for two cases: an application with a localized reaction (graffiti) and an application with a global reaction (map). Calculations of network usage for a possible binary protocol have also been included.

App	Packets/s	ASCII bytes/s	Binary bytes/s
Graffiti	68.5	90	30
Map	26	23	8

3.3 Implementation

The distributed interaction framework has been implemented using Java SE 1.6. When the framework starts running, TCP connections are established between all the PPPs. Each PPP either waits for the connection or it establishes it depending on the relation between each pair of PPP IDs. The applications have been written using the JOGL 1.1.1 library for OpenGL. This allows us to perform the geometric and photometric registrations (explained in Section 4) easily. But any library that allows linear transformations and alpha masking would work.

The camera image processing and recognition is performed in Matlab. For prototyping, Matlab is invoked to run the code, but for purposes requiring a higher performance, native code should be used. For the hand-based interaction, we use our home-grown simple Matlab-based software that detects the hand, computes its location, size and orientation, and determines its type by matching it to an existing hand posture library. However any existing 2D gesture recognition software can be used [38] for this purpose. For the laser-based interaction, a simple image thresholding detects the bright laser spot.

3.4 Results

We have prototyped four different 2D collaborative applications using this distributed interaction paradigm on our $8' \times 6'$ display wall made of a 3×3 array of nine projectors. The applications are (a) graffiti; (b) map visualization; (c) emergency management; and (d) virtual bulletin board. The graffiti application allows several users to draw at the same time with their hands. Touching the screen with two fingers brings up a palette, and the user can choose a color by tapping on it. That creates a temporary color square that the user can use to start drawing lines with that color. This window can be closed tapping on it again. The map visualization allows individual working windows (that can be moved or resized) for several users to work on different parts of the map simultaneously. The background map and the working windows can be panned and zoomed independently and the map type can be changed. The emergency management application demonstrates a possible interface that could be used to coordinate response teams in which several emergency management officials can coordinate the first responder efforts in different parts of the affected region. Markers can be added to indicate a danger area, two associated numbers indicating present and dispatched personnel can be updated, and routes can be drawn and erased to indicate the safest/fastest routes to reach or avoid danger areas. The virtual bulletin board allows the users to hang digital documents and manipulate them. Bulletins can be moved around, resized, highlighted and removed.

We have tested multi-user interaction successfully with up to five simultaneous users, but a display with more area should easily fit a much larger number of users. To demonstrate the ease of interaction we brought children ranging from ages 7 to 13 years old to draw on the display in a collaborative manner. It took them only a few minutes to get comfortable interacting with the display and its inherent scalability allowed multiple children to simultaneously draw on the screen without introducing additional latency. To demonstrate the scalability of our paradigm to different interaction modalities, we also show multi-user interaction with blue and green laser pointers with our existing interactive applications. We have shown a few static images of our applications in action in Figure 1, but we strongly encourage the readers to check the video for an accurate impression of functionality and performance.

Network usage has been measured during interaction for the cases of a gesture affecting only a few of the PPPs and for a gesture affecting all of them (Table 1). These values represent the traffic for an effective recognition refresh rate of 8.12 times per second, but no time has been spent optimizing the protocol for network usage. We have also included calculation of how much traffic there would be if a binary protocol were to be implemented.

Note that we have not explicitly assured synchronization of event execution across multiple PPPs. However, in practice we found the latency of execution of the same event across multiple PPPs to be small, less than 15ms. However, the main contributor for delay was the gesture recognition code since we used MATLAB for quick prototyping. Though this did not seem to bother our users – even the over-active children – we believe this delay should be greatly reduced using native code.

The application specific modules of the reaction manager, though non-trivial, are relatively simple to design. Adapting an existing application to our distributed interaction system took an an average graduate student one to two days. To demonstrate this, we next describe how we designed the application specific modules for the few applications we have prototyped in our lab.

To fill the *Create Event* module, the applications have to be able to decide what kind of event should be generated and define the attributes for each kind of event. For example, the Map application defines an event to pan over the map (containing the moved distance), another event to create a personal working area (containing the position and size of the created area), etc. and the Graffiti application defines an event to draw a line (containing the color and width of the line and the position of the next point), another one to open a color palette (containing the position where it should be opened), etc. In the case the same event can be applied to different objects, those attributes will contain a reference to the affected data. For example, when dragging a bulletin in the Virtual Bulletin Board application, the generated events will contain a reference number to the affected bulletin that is consistent among all the PPPs. In the *Execute Events* module, the applications apply the event depending on its type and attributes. For example, the Virtual Bulletin Board application loads a new bulletin from the data server and displays it when the event to load a bulletin is executed and the Emergency Management application displays a warning sign in the position contained in the event generated when the user does a gesture to mark a danger area.

For 'roll back', the applications have to implement a way to undo each of the events to ensure that the application returns to the exact same state as before the execution of the event. When no data is involved, this is simple and can be achieved by performing operations in the reverse order. For example, to roll back an event in the Bulletin Board application that highlighted a bulletin, the application just has to de-highlight it. However, if data is involved, we have to keep the original data before modification, since it may be impossible to get it from the modified data. Hence, all data needs to persist for a while even after it is removed from the application. The clearest example of this is when the event execution removes an object from the display and it has to be restored when undoing it (e.g., when closing a working area in the Map application).

4 Distributed Registration Improvements

In this section, we describe our new distributed registration technique in detail. We discuss related work in Section 4.1 followed by the innovations of our method in Section 4.2 and 4.3. Finally, we discuss the implementation details and results in Section 4.4.

4.1 Related Work

[3] presents a distributed registration method when using a network of $m \times n$ PPPs on a planar display. The method has three steps. (a) First, in a *neighbor detection* step a pattern made of 4 disjoint grids of 5×5 blobs (Figure 12) is used to detect the neighbors of each PPP via their cameras. (b) Next, in the *configuration identification* step binary coded information is encoded in these grids of blobs and propagated in multiple rounds via communication using the cameras to decipher the total number of PPPs in the display, their configuration (total number of rows and columns) and the coordinates of the PPP in this display. (c) Finally, in the *registration* step, the pattern in the neighbor discovery step is used to register the display using a distributed homography tree technique (See video for the method in action).

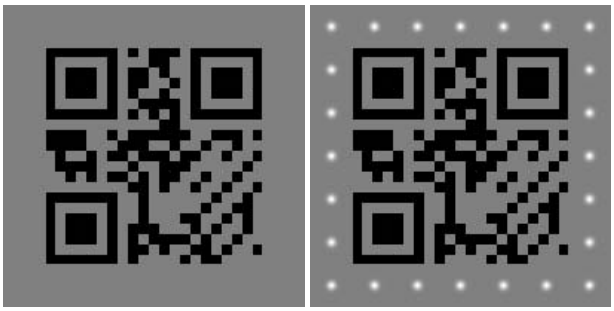

Fig. 11. Left: A standard version 1 QR Code. Right: The same QR Code augmented with our Gaussian blobs used in the registration phase.

This method has several shortcomings. First, the configuration identification step requires $O(ln(mn))$ rounds. In each round, each PPP projects its own coded pattern to communicate its belief about the display configuration (total rows and columns in the display and its own 2D coordinates in it), sees the projected coded pattern of the adjacent PPPs and changes its belief based on some rules. This continues iteratively across the PPPs until all converge to the correct configuration. However, multiple rounds of such camera based communication need considerable image processing and hence impacts performance. This also limits the scalability of the method across a larger number of PPPs. Finally, since colored patterns are used, the image processing is not robust when using commodity cameras with lower color fidelity.

Second, the homography tree technique [8] is inherently a centralized technique. A homography *graph* considers each projector as a node and places an edge between adjacent projectors i and j. Each of these edges is associated with the *local homography* between the adjacent projectors $H_{i \to j}$ recovered using the cameras. $H_{i \to j}$ is the transformation that takes pixels in the coordinate system of projector i to that of projector j. The *homography tree* technique identifies a projector P_R as the reference and finds for each projector P_i, a path to P_R. This results in a spanning tree in the homography graph, called *homography tree*, whose root is the reference projector (Figure 14). The homography that relates the projector P_i to P_R is given by the concatenation (multiplication) of the local homographies on the path from P_i to P_R in the homography tree.

The homography graph should ideally have some invariants: (a) the concatenation of homographies across a cycle in the homography graph should be identity; (b) $H_{i \to j}$ should be the inverse of $H_{j \to i}$. But this is seldom the case due to several estimation errors and small projector non-linearities resulting in significant misregistrations, especially along the edges which are not part of the homography tree. To alleviate this problem, the homography tree is usually followed by a global alignment method via non-linear optimization techniques like bundle adjustments [8]. Since any global optimization technique is hard to achieve in a distributed manner, [3] avoids this step when designing a distributed homography tree method to achieve the registration. In this distributed variant, the tree is formed in a distributed manner by communicating the homography to the reference to neighbors who choose one of the many communicated homographies and multiply it with the local homography to the neighbor to find a PPP's own homography to the reference. The process starts from the reference whose homography to itself is identity.

Comparison with our work: In our new registration technique, we introduce the following innovations. We use a single pattern based on specially augmented QR Codes to simultaneously achieve neighbor detection, configuration identification and registration. This allows us to eliminate the $O(ln(mn))$ rounds of camera based communication in the configuration identification round in [3] and achieve this in

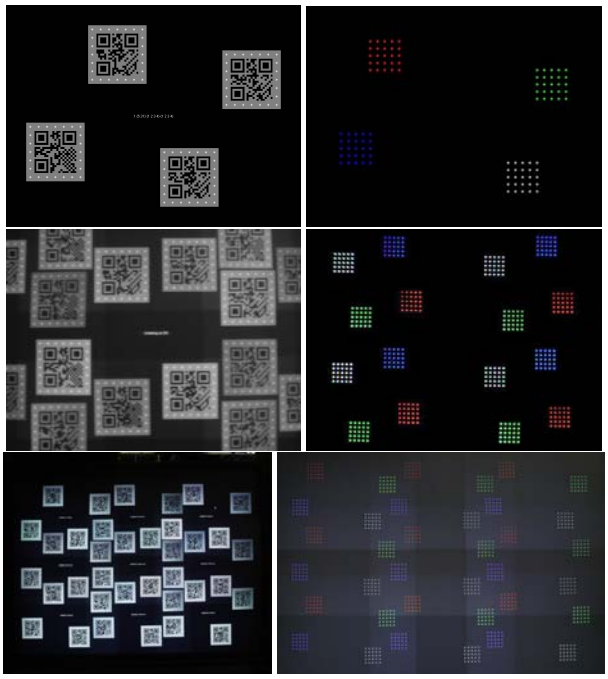

Fig. 12. We show the patterns for our work (left) compared against [3] (right). Top: The pattern projected by each PPP. Middle: The image seen by a PPP when all neighbors and itself are projecting their patterns. Bottom: The image of a 3×3 array of nine projectors projecting their patterns.

$O(1)$ time. Also, this significantly reduces visual communication iterations and image processing time thus improving performance. This faster convergence is possible by supplementing the single camera-based communication with network based communications. Hence, as opposed to a multi-second registration of [3], our registration is almost instantaneous. However, the network overhead is still at most the same as [3]. Finally, since we use monochrome patterns instead of color patterns as in [3], our image processing is much more robust and allows inexpensive cameras with much lower color fidelity.

Second, we propose a new radially cascading registration method (Section 4.3) that is amenable to a distributed implementation and achieves much superior registration. This method can also be used for any centralized tiled display and performs better than the homography based global alignment technique proposed by [26]. The homography based global alignment seeks to find a global homography G_i for each projector P_i constrained by the fact that when considering any other projector P_j, $i \neq j$, $G_j H_{i \to j}$ should provide G_i. Hence $G_i = G_j H_{i \to j}$. These provide a set of over-determined linear equations which when solved provides a transformation G_i for each projector that aligns all projectors with a common global coordinate system. This method tends to distribute the errors in the local homography across the entire display and hence, cannot remove pixel misregistration entirely. Unlike the homography tree where the misregistrations are concentrated at a few edges not included in the tree, the global alignment technique has small errors across all the overlap regions. However, the worst misregistration is significantly reduced from the homography tree technique. Our radially cascading registration method provides much superior results when compared with this global alignment technique and the distributed homography tree technique (Figure 14). The slight global distortion visible in the results of our method is due to small non-linearities in our commodity cameras.

4.2 Minimizing Camera Based Communication

QR (Quick Response) code is a 2D barcode which can embed a certain number of bits of data. The number of bits that can be encoded in the QR Code changes with its size. We encode in a QR Code the IP address and the port of the PPP, the location of the QR Code (2D coordinates of its top left corner in the projector coordinate system),

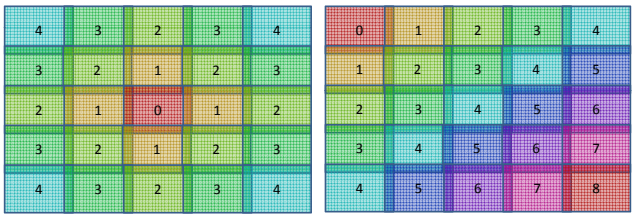

Fig. 13. Explanation of addition of PPPs to the pool of registered PPP for the radially cascading geometric registration method. Left: The middle PPP is the reference. Right: The top-left PPP is the reference.

and the size of the QR Code. The QR Code has a surrounding large 'quiet zone'. We augment the QR Code with some blobs embedded in this quiet zone which are used to detect correspondences across the projectors and the cameras for registration purposes (Figure 11. The blobs are embedded in a manner so that the quiet zone is retained after the binarization of the QR Code in the decoding phase. Hence, we can still use standard QR Code decoders without any change.

The pattern projected by each PPP as they are turned ON has four of these augmented QR Codes placed in an offset manner around the center such that each of the left, right, top and bottom neighbors can see at least one of these completely. The placement of the pattern and the overlaps required to assure no conflicts with neighboring projector are decided as in [3]. Since the camera of each PPP sees more than their own display, they see the neighbors' patterns along with their own. Figure 12 shows the pattern projected by each PPP, the image seen by each PPP and the display when all PPPs are projecting their own QR based patterns.

Each PPP detects the patterns from its neighbors to find out which of the left, right, bottom and top neighbors exist and creates the local connectivity graph of itself with its neighbors. Next, they decode this pattern to find out the exact IP-address of each of their neighbors. Finally they broadcast the location of each of their neighbors (left, right, top or bottom) along with the associated IP-address to all the PPPs in a single UDP packet. When each PPP receives this information, it augments its local connectivity graph using this information. Thus, each PPP now builds the connectivity graph for the entire display and associates a unique IP address with each node. Thus, it knows the total number of projectors in the display and their configuration. Following this, each node performs the same row-major naming convention to detect its own coordinates in the display. Unlike [3] which builds the connectivity over multiple rounds of camera based communication and broadcasts the IP addresses only following configuration identification, we achieve the same result with the same amount of network communication but with no computation overhead.

4.3 Radially Cascading Geometric Registration

Once the QR Codes are deciphered, each PPP i performs a blob detection in the quiet zone of the codes to decipher all the blobs. These blobs provide correspondences between the PPP's own projector and camera and hence allows it to recover the self-homography between its projector and camera. Next it detects the homographies with its adjacent projector j using the blobs detected in its QR Codes. Finally, it concatenates its self-homography with the homography of its camera with the adjacent projector to create the local homography $H_{i \to j}$.

The radially cascading geometric registration method starts from a reference projector which is considered as the only registered PPP initially. In each subsequent step S, PPPs with Manhattan distance S from the reference join the set of registered PPPs by aligning themselves with the PPPs who joined the registered display in step $S-1$. The process stops when all the projectors belong to the set of registered projectors. Figure 13 shows the PPPs that join the set of registered PPPs for different steps S for two reference projectors in the display, the center one and the top left one respectively. Note that for a rectangular array of PPPs, all the PPPs that join the set of registered PPPs in step S share at most two boundaries with the set of registered PPPs.

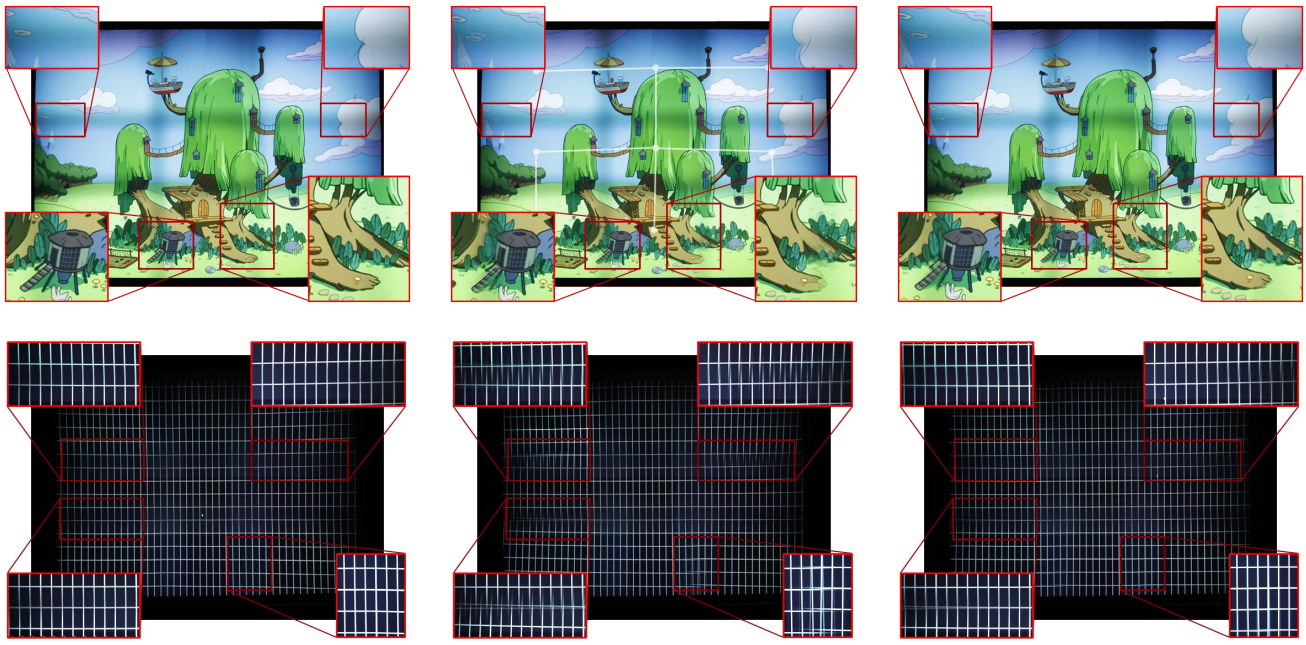

Fig. 14. Here we compare our method (top) with the homography tree (center) and the global optimization (bottom) technique on a 3×3 array of nine projectors for two different images on left and right. The homography tree (center) is shown superimposed in white – note that the misregistrations are mostly along the links which are not used in the tree. Please zoom in to see details.

In step S, all the PPPs that joined the set of registered PPPs in step $S-1$ send their homography with respect to the reference PPP to all their neighbors whose Manhattan distance is S from the reference PPP. Thus, the PPP at a Manhattan distance S recieves a homography from all the registered PPPs with Manhattan distance $S-1$ that share a boundary with it. Let us assume a PPP i in step S is receiving two such homographies from two neighbors j and k, denoted by G_j and G_k respectively. This PPP first converts the correspondences in the overlap with j to the coordinate system of the reference projector using G_j. Similarly, it converts the correspondences in the overlap with k using G_k. This gives PPP i a set of correspondences with the reference PPP via multiple possible paths through the registered projectors. PPP i then computes its own homography with the reference, G_i, using all these correspondences. This method can be summarized by a SPMD algorithm running on each PPP i as follows.

if center PPP {
    Send I to PPPs with d = 1;}
else {
    d = Manhattan distance to center;
    forall neighbors j with dist=d-1 {
        Receive G_j from j;
        Multiply all correspondences in overlap with j using G_j; }
    Estimate G_i using all correspondences with all neighbors;
    Send G_i to all neighbors j with dist=d+1; }

The total number of steps required for this algorithm to register will be the maximum Manhattan distance of any PPP from the reference. For a display of $m \times n$ PPPs, if the top left PPP is chosen as the reference, the PPP with the maximum Manhattan distance of $(m-1)+(n-1)$ from the reference is the bottom right. If the central PPP is chosen as the reference, the number of steps will be $\frac{m-1}{2}+\frac{n-1}{2}$. Figure 14 compares our superior registration with that achieved by the homography tree and the global alignment technique.

4.4 Implementation and Practical Improvements

We demonstrate our distributed calibration system on a grid of nine PPPs in a 3×3 array. Since this method does not rely on color patterns, each PPP is equipped with a monochrome VGA webcam. Due to noisy and low-resolution cameras we use the lowest resolution QR Code (203×203 pixels embedding a 29×29 grid) which can embed at most 152 bits of information. We embed a 32-bit IP address, 16-bit port, the top left corner of the code in the projector coordinate represented as two 16-bit numbers, and the 8-bit size of the code (using only 88 of the 152 bits). We embed 24 Gaussian blobs in the quiet zones of the QR Codes. Gaussian blobs allow us to robustly determine blob positions with subpixel accuracy and improve the quality of homography estimation. We use the ZBar barcode library to quickly decode QR Codes seen by our cameras as well as provide a rough estimate of the QR Code corners.

Since each PPP independently builds the complete graph of the display, our radially cascading geometric registration technique can be performed either via message passing (Section 4.3) or independently on each PPP after it forms the adjacency graph for the entire display reducing the network overhead significantly. Hence, we include the PPP's local homographies with all its neighbors in the message broadcast during the configuration identification stage. Each PPP i sees only one of the four QR Codes for its neighbor j completely. The four corners of the QR Code are used to estimate a coarse local homography, which is used to initialize the blob detector. The detected blob positions in this QR Code are then used to produce a more accurate local homography estimate. For each projector i, we find the two homographies with each of its neighboring PPP j, $H^i_{i\to j}$ and $H^i_{j\to i}$, where the superscript denotes the PPP which computes these homographies. Note that the same homographies can be computed by j as well, $H^j_{i\to j}$ and $H^j_{j\to i}$ respectively. Ideally, $H^i_{i\to j} = H^j_{i\to j}$. But, due to the distribution of the blobs only around the QR Codes instead of the entire overlap, especially in the face of mild non-linearities in either the projector or camera, this results in a situation where there can be a slight deviation from this constraint. So, we design a method to compute a more accurate homography $H_{i\to j}$ from i to j, as follows. We generate a set of points uniformly over the overlap of i with j and find their corresponding points in j using $H^i_{i\to j}$. Similarly, we generate a set of points uniformly over the overlap of j with i and find their corresponding points in i using $H^j_{i\to j}$. Then we use this combined set of correspondences to generate a more robust estimate of $H_{i\to j}$ using a linear least squares technique. In our system this computation of the radially

cascading registration on each PPP did not exceed the network latency and time to sequentially compute and propagate this information.

To achieve photometric seamlessness, we use the recovered homography of each PPP i with its neighbor j to detect the exact shape of the overlap. Finally, each PPP independently applies a blending function in each of its overlap.

5 Conclusion

In conclusion, we present the first distributed interaction paradigm for large rear projected display walls. We demonstrate the scalability of our method to multiple displays, users, applications and interaction modalities by showing a working prototype of multiple 2D applications using both gestures and laser based interaction modality. We also propose a new distributed registration technique that is more accurate and efficient than prior methods. This technique is deterministic, can be easily implemented for a centralized system, and does not involve time-consuming global optimizations.

In the future we would like to extend our work for front projection systems where occlusion is an issue. We would like to extend it beyond 2D applications. We believe that our paradigm can extend to 3D applications, however we would like to explore the different issues in detail. We would also explore designing distributed versions of the more rigorous photometric calibration methods like [22, 28].

References

[1] S. C. Ahn, T.-S. Lee, I.-J. Kim, Y.-M. Kwon, and H.-G. Kim. Large display interaction using video avatar and hand gesture recognition. *International Conference on Image Analysis and Recognition*, pages 261–268, 2004.

[2] E. Bhasker, R. Juang, and A. Majumder. Registration techniques for using imperfect and partially calibrated devices in planar multi-projector displays. *IEEE TVCG*, pages 1368–1375, 2007.

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

[4] M. S. Brown and B. W. Seales. A practical and flexible tiled display system. *IEEE Pacific Graphics*, 2002.

[5] X. Cao and R. Balakrishnan. Visionwand: Interaction techniques for large displays using a passive wand tracked in 3d. *ACM symposium on User interface software and technology*, pages 193–202, 2003.

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

[7] 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.

[8] H. Chen, R. Sukthankar, G. Wallace, and K. Li. Scalable alignment of large-format multi-projector displays using camera homography trees. *Proc. of IEEE Vis*, pages 339–346, 2002.

[9] J. Davis and X. Chen. Lumipoint: multi-user laser-based interaction on large tiled displays. *Displays*, 23:205–211, 2002.

[10] P. Dietz and D. Leigh. Diamondtouch: A multi-user touch technology. *ACM symposium on User interface software and technology*, pages 219–226, 2001.

[11] T. Grossman, R. Balakrishnan, G. Kurtenbach, G. W. Fitzmaurice, A. Khan, and W. Buxton. Interaction techniques for 3d modeling on large displays. *ACM Symposium on Interactive 3D Graphics*, pages 17–23, 2001.

[12] J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. *ACM Symposium on User Interface Software and Technology*, pages 115–118, 2005.

[13] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan. Distributed rendering for scalable displays. *Proceedings of IEEE Supercomputing*, pages 129–140, 2000.

[14] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl: A scalable graphics system for clusters. *Proceedings of ACM SIGGRAPH*, pages 129–140, 2001.

[15] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahem, P. Kirchner, and J. Klosowski. Chromium : A stream processing framework for interactive rendering on clusters. *ACM Transactions on Graphics (SIGGRAPH)*, pages 693–702, 2002.

[16] S. Izadi, H. Brignull, T. Rodden, Y. Rogers, and M. Underwood. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. In *ACM symposium on User interface software and technology*, pages 159–168, 2003.

[17] H. Jiang, E. Ofek, N. Moraveji, and Y. Shi. Direct pointer: direct manipulation for large-display interaction using handheld cameras. In *SIGCHI conference on Human Factors in computing systems*, pages 1107–1110, 2006.

[18] A. Khan, G. Fitzmaurice, D. Almeida, N. Burtnyk, and G. Kurtenbach. A remote control interface for large displays. *ACM symposium on User interface software and technology*, pages 127–136, 2004.

[19] J. Kim, J. Park, H. Kim, and C. Lee. Hci(human computer interaction) using multi-touch tabletop display. *Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 391–394, 2007.

[20] C. Krumbholz, J. Leigh, A. Johnson, L. Renambot, and R. Kooima. Lambda table: High resolution tiled display table for interacting with large visualizations. *Workshop for Advanced Collaborative Environments (WACE)*, 2005.

[21] A. Majumder and R. Stevens. Color nonuniformity in projection-based displays: Analysis and solutions. *IEEE Transactions on Vis and Computer Graphics*, 10(2):177–188, March–April 2003.

[22] A. Majumder and R. Stevens. Perceptual photometric seamlessness in tiled projection-based displays. *ACM TOG*, pages 111–134, 2005.

[23] S. Malik, A. Ranjan, and R. Balakrishnan. Interacting with large displays from a distance with vision-tracked multi-finger gestural input. *ACM symposium on User interface software and technology*, pages 43–52, 2005.

[24] D. R. Olsen and T. Nielsen. Laser pointer interaction. *ACM conference on human factors in computing systems*, pages 17–22, 2006.

[25] R. Raskar. Immersive planar displays using roughly aligned projectors. In *Proc. of IEEE VR*, pages 109–115, 2000.

[26] R. Raskar, J. van Baar, P. Beardsley, T. Willwacher, S. Rao, and C. Forlines. ilamps: Geometrically aware and self-configuring projectors. *ACM Transaction on Graphics (SIGGRAPH)*, pages 809–818, 2003.

[27] J. Rekimoto. Smartskin: An infrastructure for freehand manipulation on interactive surfaces. *ACM conference on human factors in computing systems*, 2002.

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

[29] C. Shen, F. D. Vernier, C. Forlines, and M. Ringel. Diamondspin: an extensible toolkit for around-the-table interaction. In *SIGCHI conference on Human factors in computing systems*, pages 167–174, 2004.

[30] D. Stødle, J. M. Bjørndalen, and O. J. Anshus. A system for hybrid vision- and sound-based interaction with distal and proximal targets on wall-sized, high-resolution tiled displays. *IEEE International Workshop on Human Computer Interaction*, pages 59–68, 2007.

[31] D. Stødle, Tor-Magne, S. Hagen, J. M. Bjørndalen, and O. J. Anshus. Gesture-based, touch-free multi-user gaming on wall-sized, high-resolution tiled displays. *4th International Symposium on Pervasive Gaming Applications, PerGames*, pages 75–83, 2007.

[32] D. Stødle, O. Troyanskaya, K. Li, and O. J. Anshus. Device-free interaction spaces. *IEEE Symposium on 3D User Interfaces*, pages 39–42, 2009.

[33] B. Ullmer and H. Ishii. The metadesk: Models and prototypes for tangible user interfaces. *ACM symposium on User interface software and technology*, pages 223–232, 1997.

[34] D. Vogel and R. Balakrishnan. Distant freehand pointing and clicking on very large high resolution displays. *ACM symposium on User interface software and technology*, pages 33–42, 2005.

[35] A. D. Wilson. Touchlight: An imaging touch screen and display for gesture-based interaction. *International Conference on Multimodal Interfaces*, pages 69–76, 2004.

[36] C.-O. Wong, D. Kyoung, and K. Jung. Adaptive context aware attentive interaction in large tiled display. *IEEE International Workshop on Human Computer Interaction*, pages 1016–1025, 2007.

[37] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. *ACM Symposium on User Interface Software and Technology*, pages 193–202, 2003.

[38] Y. Wu and T. S. Huang. Vision-based gesture recognition: A review. In *GW '99: Proceedings of the International Gesture Workshop on Gesture-Based Communication in Human-Computer Interaction*, pages 103–115, 1999.