Asynchronous Distributed Calibration for Scalable Reconfigurable Multi-Projector Displays

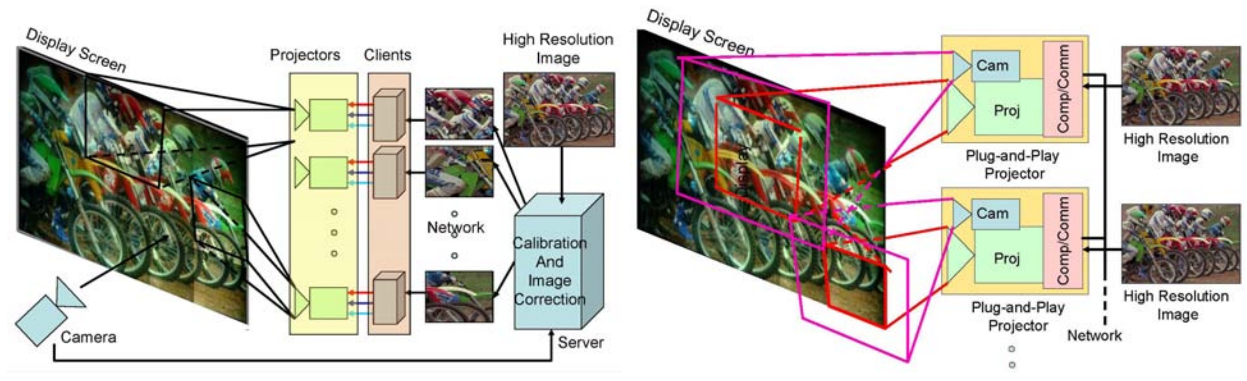


Figure 1: Centralized (left) and distributed (right) system architecture for multi-projector displays

Abstract

Centralized techniques have been used until now when automatically calibrating (both geometrically and photometrically) large high-resolution displays created by tiling multiple projectors in a 2D array. A centralized server managed all the projectors and also the camera(s) used to calibrate the display. In this paper, we propose an *asynchronous distributed calibration* methodology via a display unit called the *plug-and-play projector (PPP)*. The PPP consists of a projector, camera, computation and communication unit, thus creating a self-sufficient module that enables an asynchronous distributed architecture for multi-projector displays. We present a single-program-multiple-data (SPMD) calibration algorithm that runs on each PPP and achieves truly scalable and reconfigurable displays without any input from the user and instruments novel capabilities like *adding/removing* PPPs from the display dynamically, *detecting faults*, and *reshaping* the display to a reasonable rectangular shape to react to the addition/removal/faults. To the best of our knowledge, this is the first attempt to realize a completely asynchronous and distributed calibration architecture and methodology for multi-projector displays.

1 Introduction

Large high-resolution displays created by tiling multiple display units in a 2D array [25, 6, 18, 10] are used regularly for many applications like visualization, training, simulation and collaboration. Projectors are usually preferred over LCD panels in such applications since the mullions bordering the LCD panels make them incapable of generating one seamless image. However, projection based tiled displays suffer from two other problems (Figure 3d). (a) The image is not geometrically matched across the projector boundaries. (b) The color and brightness of the image is non-uniform due to overlap in the projected area of adjacent projectors on the screen which appears doubly bright and also varying color/brightness within and across projectors.

In the early days of tiled displays, prohibitive cost of projectors and driving engines limited the number of projectors to a handful allowing manual geometric alignment and color balancing of the display. With the advent of commodity projectors and PC clusters to drive them, displays with a large number of projectors are very affordable. But, manual calibration methods are both infeasible and unscalable for such large displays. So, several camera-based calibration techniques have been devised to calibrate these displays automatically, repeatably and inexpensively [19, 24, 28, 11, 3, 26, 4, 21, 20, 29].

Until now, all camera-based calibration techniques have a *centralized architecture* where one central machine/process bears the sole responsibility of achieving the geometric/color calibration by capturing specific projected patterns using a camera, analyzing them to generate the correction parameters, applying correction to different parts of the image to compensate for each projector's unique geometric and color artifacts, and finally shipping these images to the projectors to create a seamless display (Figure 1). The advantage of centralized calibration is in having a common global reference frame to address the pixel geometry and color. Thus, managing multiple display units to create a global seamless image is relatively easy.

However, centralized calibration is not scalable (increasing the number of projectors making up the display) or reconfigurable (changing the shape, aspect ratio and resolution of the display). Further, it is intolerant to faults, especially in the central server. In addition, deploying a centralized multi-projector display demands an educated user to set up the computers, projectors and camera appropriately, input the right parameters to the central server and maintain the whole set-up periodically.

Projectors today are affordable. Thus, building mammoth displays of billion pixels by tiling hundreds of projectors is not unthinkable. At the other end of the spectrum, smaller, mobile and flexible "pack-and-go" displays are very much desired for applications like map and troop-movement visualization in the war ground. They can even be used in public venues like schools and museums. A centralized calibration architecture inhibits the realization of the full potential of using projectors in these kinds of scenarios.

1.1 Main Contribution

In this paper we propose *asynchronous distributed calibration* of multi-projector displays where no central process/computer needs to know the number of projectors, their configuration, and the geometric/photometric relationship between the projectors *a priori* to create a seamless display. Our main contributions in this paper are the following.

1. First, we identify the minimal self-sufficient unit to realize the distributed asynchronous calibration - a *plug-and-play projector (PPP)*. This consists of a projector, camera, computation and communication unit.

2. We formalize the architecture and capabilities of a display realized by the PPPs via a distributed asynchronous calibration process.

3. Finally, we propose an *asynchronous distributed methodology* that includes methods to (1) detect the number of neighbors of a PPP, (2) find the position of a PPP in a large display, (3) achieve geometric calibration and photometric blending in a distributed manner, (4) add and remove projectors from the display dynamically for flexibility, and (4) tolerate faults for robustness. Unique to these methods is the importance of *camera-based communication* - the use of visual feedback from the PPP's camera as a mode of communication.

Our distributed asynchronous calibration via plug-and-play projectors can be instrumental in using projectors for building mobile, flexible and easily deployable displays. One can imagine a layman user creating a display by just setting a few PPPs side by side without worrying anything about calibrating them. He can also rearrange the PPPs in a different configuration or add/remove PPPs to the existing configuration to create a display of different aspect ratio without bothering about calibration. The PPPs self-calibrate to create a seamless imagery in all scenarios. In fact, this can spark and foster new paradigms of interaction, especially for collaboration and visualization. Each person can carry his own plug-and-play projector since they are cost-effective and light-weight. When more than one person meet for collaboration, their respective devices can be put together to create a seamless tiled display. Even in display walls made of a large number of PPPs, when one or more PPPs fail, the PPPs can automatically reconfigure, recalibrate and function at a limited capability (lower resolution). In summary, our work enables *self-calibrating tiled displays* liberating the user from any responsibility of set-up or maintenance.

In the rest of the paper, we first give an overview of our system (Section 2). Then we describe the basic distributed methodology that would run asynchronously on the PPPs making up a multi-projector display, followed by description of advanced features of addition/removal of PPPs and fault tolerance (Section 3). We conclude by discussing the potential of our distributed calibration in realizing ubiquitous pixels (Section 4).

2 System Overview

In this section we give an overview of the system and compare it with existing work. We first describe the plug-and-play projector, followed by the distributed calibration architecture, capabilities and assumptions.

2.1 Plug and Play Projectors (PPP)

Pixels, being the generalized purveyors of information, have been a critical commodity in any workspace with several functionalities including collaboration, visualization and interface. The particular form of pixels provided by projectors, i.e. photons cast onto an arbitrary surface from a distance, provides a unique flexibility and mobility to this useful commodity by liberating them from the spatial constraints imposed by other displays like CRT or LCD panels [25, 28]. However, when used in isolation, projectors act like passive digital illumination devices, pixels from which cannot provide us the desired high-end functionality, flexibility and mobility desired in a workspace. So, many researchers have proposed a marriage between projectors and cameras to provide 'intelligent' pixels [26, 23, 2, 5, 1]. The display unit we propose next is largely inspired by such previous works.

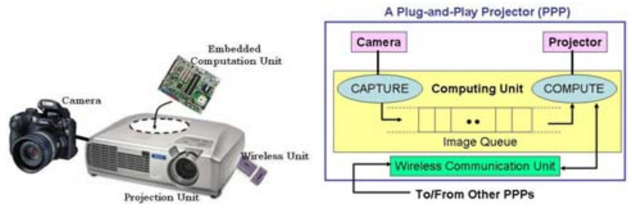


Figure 2: Left: A Plug-and-Play Projector (PPP); Right: The processes on a PPP

We propose a display unit consisting of a projector, a camera, and an embedded computing and communication hardware (Figure 2). We call this networked and intelligent projector, the *plug-and-play projector (PPP)*. Each PPP thus is a self-sufficient unit with the capacity to sense environmental changes (through the camera), adapt/react to those changes (through the computation unit) and share those changes with other PPPs if required (through the communication unit). We envision the use of these units like 'plug-and-play' lego pieces to create a scalable and reconfigurable display. Though inspired by other previous work, our PPP has some novel features. First, unlike previous smart projectors, the projector display and camera capture processes in our PPP need not be synchronized with each other. Second, this is the first time the complete potential of camera-based communications is harnessed, not only to calibrate but even to decide the position/configuration/neighborhood of each PPP in the display enabling a self-calibrating tiled display.

2.2 Architecture

In a distributed architecture, we expect every PPP to take complete control of the part of the display it is responsible for (Figure 1). Each PPP can act like a client and request the appropriate part of the data from a traditional data server (centralized or distributed) that is oblivious of the fact that the clients (the PPPs) requesting data are in reality display units. The PPPs are treated just like any other data requesting client. Thus distributed methodologies are used in all aspects of multi-projector display including calibration, data handling and rendering.

Existing distributed rendering architectures like Chromium [12, 13] and SAGE [27] use distributed methodologies only for rendering the pixels and use centralized architecture for calibration and data handling. This is the essential difference between these systems and our proposed system. In Chromium or SAGE, the user defines in the central server, the total number of display units and the relationship of the image projected by each unit with respect to the large image they are creating together. The centralized unit finally streams the appropriate data to the dumb display units or projectors.

The second advantage of our proposed system is asynchronous communication between different PPPs and also between the capture and projecting processes within the same PPP. This enables each PPP to start off as a lone unit in the display and then discover other PPPs in the environment and their configurations. However, note that during display of application data synchronization of the rendered frames are necessary. Since each display unit is just like a standard data requesting client, standard distributed synchronization approaches can be used for this synchronization.

Capabilities: Our proposed distributed asynchronous architecture provide the following capabilities.
(a) The PPPs calibrate themselves geometrically and photometrically to create a seamless imagery without any input from the user – neither the total number of PPPs making up the display and their configurations (number of rows and columns) nor the number neighbors each PPP have and their identities.
(b) Any PPP can be added to or removed from the pool of PPPs dynamically to scale/reshape/reconfigure the display.
(c) In case of faults in PPPs, the display reconfigures itself automatically to a reasonable shape.

Assumptions: Following are the assumptions we make about the projector-camera setup in the PPP.
(a) The camera of the PPP has a wider field-of-view(FOV) than the projector. Thus, the camera can see the image projected by its projector completely and also parts of the images projected by the neighboring PPPs. We assume that the camera can see about $\frac{1}{3}$ of the FOV of the projectors in neighboring PPPs.
(b) Since each camera captures a small part of the display, we use a low-resolution (640×480) inexpensive VGA video camera.
(c) The camera and projector coordinate systems of a PPP and its neighbors are rotated by less than 45 degrees with respect to each other, a reasonable assumption for a rectangular array of PPPs on a planar display.
(d) The cameras need not be synchronized with the projector in the PPP. However, following Nyquist sampling criteria, the frame rate of the camera should be double of that of the projector to assure that a pattern projected by the PPP or the adjacent PPP is not missed by the camera. If this criterion is not satisfied, projectors have to project their calibration patterns for more than one frame during calibration.

Role of Camera Based Communication: Since each PPP can sense changes in its neighbors via its own camera, a sensor/camera based communication mode is established between adjacent PPPs via analysis of the captured image. This communication channel enables critical capabilities like discovering local topology of the PPPs in the display array and detecting addition/removal of neighboring PPPs.

Camera-based communication can be used to communicate any information with no network overhead and hence can completely replace network-based communication channel. However, camera based communication is compute intensive. So, we retain a low bandwidth wireless communication unit on each PPP to allow network-based communication for tasks which would otherwise need complex compute-intensive image processing. Thus, we use the network-based and camera-based communication effectively to balance the *compute* and *network* resources of the system.

3 Asynchronous Distributed Calibration

The asynchronous distributed calibration methodology follows a single-program multiple-data (SPMD) model in which every PPP executes the exact same program. In this section we describe our method for a display made of a fixed number of projectors. We augment this basic algorithm with more advanced capabilities in Section 3.4.

Each PPP runs two asynchronous processes, *Capture* and *Compute* (Figure 2), that communicate via a shared *queue* of images, Q, and a shared boolean *CALIB*. The *Capture* process captures images from the camera and enqueues them in Q. It only stores the images that reflect a change when compared to the last image enqueued in Q. The *Compute* process dequeues images from Q, analyzes them, and computes different configuration and calibration parameters. It is assumed that when Compute process sends an image to be displayed on the projector, it keeps it projected unless asked to display another one. *CALIB* is used to denote the state of the PPP. A PPP can be in two states, (a) the *calibration state* when it calibrates itself and (b) the *stable state* when it projects application data on the display. Capture and Compute thus act like Producer-Consumer processes traditionally used in distributed computing environment, assuring mutual exclusion while accessing shared data structures.

Following are the SPMD algorithms for these processes. Parts of these programs handle addition/removal of PPPs and faults that are explained in Section 3.4.

```
Algorithm Process-Capture
Image-Queue $Q$;
Boolean $CALIB$;
begin
1.  $I$ = Capture Image from Camera;
2.  if $CALIB$ then
3.      if ($I$ is different from previous image in $Q$) then
4.          Enqueue $I$ in $Q$;
5.      endif;
6.  else
7.          Addition-Removal-Handling-for-Capture
8.  endif;
end
```

```
Algorithm Process-Compute
Image-Queue $Q$;
Boolean $CALIB$;
begin
1.  CALIB = True;
2.  if (CALIB) then
3.      Neighor Discovery;
4.      Configuration Identification;
5.      Photometric Blending;
6.      Geometric Alignment;
7.  else // Stable state
8.      Request Data from Server;
9.      Correct and Project Data;
10.     Addition-Removal-Handling-for-Compute
11. endif;
end
```

The *Compute* process is the core of our method. Our asynchronous distributed method starts running as soon as

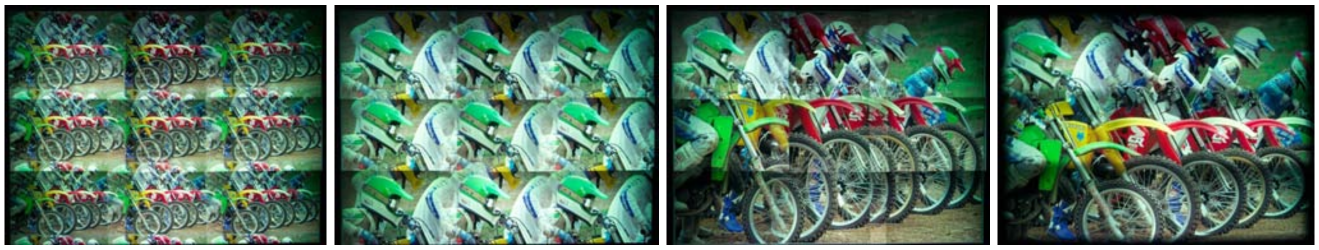

(a) (b) (c) (d)

Figure 3: A multi-projector display made of 9 PPPs, if allowed to project data (a) at power on would display the entire data since it thinks it is lone display unit responsible for data display; (b) at the beginning of Configuration Identification process when it is aware that other PPPs would share the responsibility of data display, but thinks that it's position is $(1,1)$ and would display the first 1024×768 part of the image; (c) after Configuration Identification step would display the right part of the image, but geometrically and photometrically uncalibrated; and (d) finally, after Alignment would project the seamless image. (d) is the only image that the user would see, rest are used for illustration.

the projector is powered on and involves three steps. Figure 3 illustrates *the perception of each PPP* about the presence of other PPPs in the environment and its configuration at the end of each of these steps.

1.*Neighbor Discovery:* In this stage each PPP checks for the existence of left, right, top or bottom neighbor using camera-based communication with adjacent PPPs.

2.*Configuration Identification:* Next, the PPPs find the dimensions of the display array, their own coordinates in the array, and the IP addresses of all the PPPs in the display. Camera-based communication between adjacent PPPs and a network broadcast is used for this purpose.

3.*Alignment:* In this step, a seamless image is generated by calibrating the display geometrically and photometrically using network-based communication. For geometric alignment a distributed homography tree technique is used. Intensity blending is used in the overlapping regions to achieve photometric seamlessness.

3.1 Neighbor Discovery

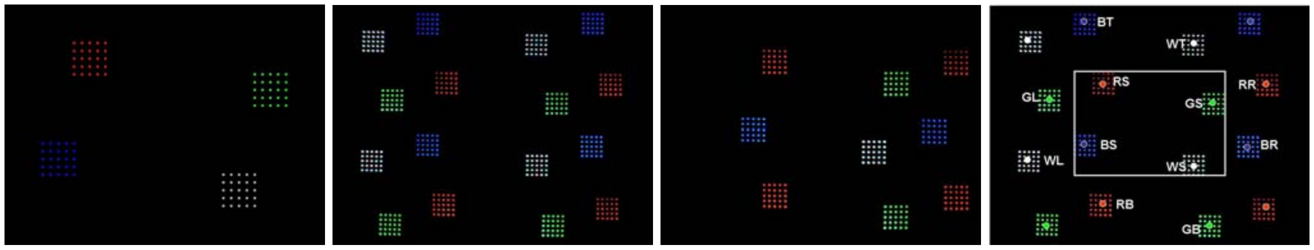
The neighbor discovery starts with the assumption that none of the four neighbors (left, right, top or bottom) of a PPP exist. Each PPP projects a pattern consisting of clusters of colored blobs. The image captured by the camera of a PPP contains clusters from its own and part of the neighbor's projected pattern. The patterns are designed such that the colors and locations of the clusters can be analyzed to detect the presence/absence of the four neighbors. Following is the detailed description of the pattern used and the algorithm.

The Pattern: The pattern contains four *clusters*, each made of 5x5 array of *square blobs* (Figure 4). The locations of these clusters are optimized to avoid overlap of the clusters projected by adjacent PPPs and determines the maximum allowable overlap between adjacent PPPs. Our pattern allows a maximum of 200 pixels overlap (20% of the projector's resolution). Each cluster has a different color - red, green, blue or white respectively in top-left, top-right, bottom-left and bottom-right corners of the image. This allows a PPP to differentiate the clusters projected by itself from those projected by its neighbors. Following is the algorithm that generates the neighborhood information (Pseudocode in Appendix).

Identifying Location of the Clusters: First, we identify the centers of the blobs using standard blob detection techniques [24, 11]. Next, we spatially group the detected blobs into an array of clusters. We use a hierarchical agglomerative clustering approach [8] that has time complexity of $O(n^2 ln(m))$, where n is the total number of blobs in the image and m is the dimension of each cluster (5 in our case). This method does not need as input the total number of blobs present in the image and hence can handle asynchronous display of patterns from adjacent PPPs.

Identifying Color of the Clusters: Next, the color of each cluster is determined in three steps.
(a) First, the color of each blob is detected by applying chromatic filters centered at the blob centers. The detected colors, being in the camera's color space may not coincide with the projected colors due to variations between the camera and projector color gamuts. So, we assign to each blob the projected color that has the minimum angular deviation with the detected color in RGB space.
(b) Due to small gamut variations across the projector and camera, all the blobs in a cluster may not be assigned the same color. So, cluster colors are determined by majority voting of the colors of the component blobs.
(c) Next, each PPP creates a *Chromatic Blob Table (CBT)* for *each color*. The CBT lists the centers of all detected blobs in the camera coordinate along with the *id, color, and center* of the *cluster* they belong to.

Identifying Owners of the Clusters: Any cluster detected by the PPP in the previous step either belongs to itself or to its adjacent PPP. We identify the owner of each cluster using the following three steps. We consider four connected neighbors and do all computations on the centers of clusters.
(a) Since each PPP is guaranteed to see its own pattern before or along with the pattern of adjacent PPPs, first an axis-aligned bounding rectangle enclosing the PPPs own clusters is deciphered (Figure 4d). This can be done by finding the minimum x from the red and blue, maximum x from the green and white, maximum y from the blue and white and minimum y from the red and green CBTs.
(b) Next each cluster is assigned its closest corner in the rectangle.
(c) Finally, based on the color of the cluster and its associated rectangle corner, the ownership of the cluster is resolved. For example, three clusters RGW associated with the top right corner of the rectangle belongs to the right neighbor, self, and top neighbor respectively. Figure 4 shows the centers of the chromatically classified clusters and labels these centers to denote their ownership. The first letter of the labeling denotes the color (RGBW) and second letter denotes the ownership (S for self, and LRBT for its four neighbors). The CBT is also updated to include the owner of each cluster. Further, for each PPP, its neighborhood information is also resolved during this process.



(a) (b) (c) (d)

Figure 4: (a) The pattern projected by the projector of a PPP in Neighbor Discovery step. (b) The pattern seen by a camera of a PPP when all its adjacent projectors including itself in projecting the pattern in (a). (c) The pattern seen by a camera of a PPP when only two of its neighbors and itself are projecting the pattern. Due to the asynchronous nature of the system, there is no guarantee that all the PPPs project their pattern at the same time and this situation is likely to occur.

3.2 Configuration Identification

In this configuration identification step, each PPP finds the display array dimension, its own coordinate in it and the IP addresses of all the PPPs in the display. Binary coded bit patterns embedded in the cluster of blobs are used to convey each PPP's beliefs about where it is in the display, and the total dimensions of the display. Every PPP starts by believing that it is the only node in a display of dimension 1 by 1. Multiple rounds of camera-based communication between adjacent projectors follow when each PPP updates its own row(r) and column(c), and the total rows(m) and columns(n). Update rules enable *propagation* of these parameters to all the PPPs in the display. This results in convergence to the correct configuration at each PPP. This is followed by a network-based communication step where each PPP gathers the IP address of all other PPPs in the display. Following is the detailed description of the pattern and process (Pseudocode in Appendix).

Pattern: The pattern for this step is derived by binary encoding all the clusters similarly. Each blob denotes 1 when off and 0 when on. The first two rows are used to encode the row r and column c of the PPP, and the third and fourth rows are used to encode the total number of rows and columns, m and n respectively. The last row is used to denote five *status bits* - the first four bits denote if r, c, m and n have converged to the final correct values on this PPP. The final bit *isdone* is turned on when all of the four status bits are set to denote the completion of the configuration identification process on this PPP. Figure 5 shows an example of the binary encoding.

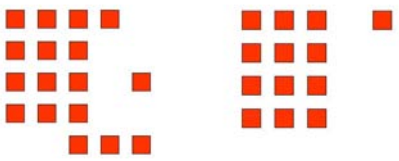


Figure 5: The binary encoded grids with $(r,c) = (1,3)$, $(m,n) = (2,3)$, $(r_s,c_s) = (True,True)$ and $(m_s,n_s,isdone) = (False,False,False)$ (left) and $(r,c) = (2,3)$, $(m,n) = (3,3)$, $(r_s,c_s) = (True,True)$ and $(m_s,n_s,isdone) = (Ture,True,True)$ (right).

Finding display dimension and PPP coordinate: The configuration identification starts by each PPP setting their (r,c) and (m,n) to $(1,1)$ and all its status bits to $False$. Then, each PPP performs the following steps in an iterative manner till *isdone* is set indicating its convergence to the correct values of (r,c) and (m,n).
(a) First, the PPPs capture the image of the encoded bit patterns and decipher the (r,c), (m,n) and the status bits of the neighbors from the image captured by the camera. This is done by analyzing the presence or absence of blobs at the blob centers stored in the CBTs.
(b) Next, this information is used to update its own (r,c) and (m,n) and status bits following some *update rules* (Algorithm *Update-IDs* in Appendix).
(c) Finally, the PPP changes the embedded binary coding in its cluster and projects the new image.

The above steps on each PPP result in *propagation* of the values of the encoded parameters in the following way.
(a) The PPP with no left and top neighbor (top-left PPP in the array) initiates the process and indicates that its (r,c) of $(1,1)$ has converged by setting appropriate status bits.
(b) Each PPP updates its (r,c) parameters from the *top or left* neighbor. This process continues and the row and column changes propagate from the top left of the display to the bottom right in a breadth-first manner where PPPs in the same level of the tree perform the updates in parallel. This *front propagation* of (r,c) completes in $O(ln(mn))$ steps.
(c)When the bottom right projector detects convergence of its (r,c) parameter, it sets the (m,n) to be the same as its (r,c) and turns on its *isdone* status bit to indicate convergence to the correct configuration parameters.
(d)Each PPP now updates their (m,n) and *isdone* from their *bottom or right* neighbor, leading to a *back propagation* of parameters from the bottom-right to top-left of the display, again in a breadth-first manner in $O(ln(mn))$ steps.
Thus, each PPP discovers the correct configuration parameters using only camera-based communication between adjacent projectors.

Finding IP addresses of all PPPs: Next, each PPP broadcasts its coordinates in the display along with their associated IP address over the network. On receiving this broadcast message, each PPP creates a table that maintains coordinate of every PPP in the display along with the associated IP addresses. This step enables network communication between adjacent PPPs during alignment. Note that this step can be done before discovering the total number of display units and their configurations and via a standard communication protocol like UPnP or Apple's Bonjour/ex-Rendez-Vous/ZeroConf. However, using a standard protocol does not allow a seamless integration of this step with the rest of the calibration methodology.

Handling Race Conditions: In an asynchronous system, it is possible that a neighbor of a PPP is performing its neighbor discovery step while the PPP is in its configuration identification step. This situation is detected by identifying appearances of new blobs in the captured image that are not present in the CBT. To handle this race condition, the PPP *aborts* its current step and goes back to the neighbor discovery step where it lets its neighbor know of its presence. It indicates this abortion by turning off its convergence bits which enables propagation of this information to other non-adjacent PPPs. This also allows all the PPPs in the display to stall their convergence until information from the new PPP propagates.

3.3 Alignment

In this step the PPPs find their exact geometric relationship with each other (amount of overlap, relative alignment of images) and use it for geometric alignment and photometric blending. First, each PPP uses the Hungarian method to detect correspondence between the blobs in the CBTs with those in the projected pattern and computes the local homography with each neighbor. This, in turn, is used to compute the overlap with its neighbors and blend it photometrically. To align the images geometrically, we design a distributed homography tree technique that aligns the images from all PPPs with respect to one reference PPP to achieve a seamless display. Following is a detailed description of the process. (Pseudocode in Appendix).

Correspondence Detection Using Hungarian Method: Previous methods would correspond detected blobs in the camera space with the projected blobs in two different ways. (a)[24, 28] binary code the blobs and project them in a time sequential manner. (b) [11, 3] project all blobs in one frame and then determine some distance parameters to walk along the blobs in a scan-line order in the projector coordinates. Usually additional patterns are projected to calibrate these parameters. Both methods use multiple patterns for correspondence detection. In an asynchronous distributed system as ours, tracking multiple frames from each PPP is not viable. So, we devise a novel way to detect correspondences using the Hungarian method.

The Hungarian method is a strongly-polynomial combinatorial optimization algorithm due to Kuhn [17] and later revised by Munkres [22]. It is used to solve bipartite matching (i.e., the assignment problem). The spatial clustering that generates the clusters in the neighbor discovery stage does not impose any order to the clustered blobs. To find the order, we generate a generic template by finding the axis-aligned bounding rectangle for the detected cluster and populating it with 5×5 array of blobs. Then, a cost matrix is computed by taking the Euclidean distance between every detected-blob and template-blob pairs. Each element represents the "error" induced when suggesting that particular assignment. The Hungarian algorithm then operates on this matrix to find the assignment of detected-blobs to template-blobs that minimizes total assignment error (the sum of the square of all distances). Thus, we order the blobs robustly and automatically. From the known order and color of the blobs we can find the exact correspondence of blobs between the camera and projector coordinates.

The Hungarian method however is applicable only in scenarios where the cameras and projectors of adjacent PPPs are rotated by less than 45 degrees with respect to each other. This is a reasonable assumption for rectangular display. However, for more general arrangements of projectors, the color coding of the clusters can be used to provide more information to this method where larger relative angles between the cameras can also be tolerated.

Local Homography Calculation: Next, the correspondences are used to calculate homographies, a linear relationship tying the different device coordinates. Let the projector of a PPP be denoted by P and the camera by C. The homography between two devices A and B is denoted by $H(A \rightarrow B)$. Each PPP calculates the *self homography* relating the PPPs own projector and camera, $H(C_{r,c} \rightarrow P_{r,c})$. It also computes the *local homography* to each neighbor, $H(P_{(r+k),(c+k)} \rightarrow P_{r,c})$ where $k \in \{-1, 0, 1\}$, as $H(P_{(r+k),(c+k)} \rightarrow C_{r,c}) * H(C_{r,c} \rightarrow P_{r,c})$.

Local Photometric Blending: Using the local homography each projector finds the overlap with its neighbor and applies a linear or cosine blending (in the horizontal or vertical direction) to the RGB colors in this region [29].

Aligning Geometry Distributedly: We design a distributed methodology for the homography tree technique [3] to achieve geometric alignment. This starts with election of a PPP close to the center of the display as the root of the homography tree. All other PPPs align themselves with respect to the root. The homography tree is built in a breadth-first fashion in $O(ln(mn))$ steps using network-based communication across adjacent PPPs. The process is initiated by the root. Each PPP sends its homography-to-root to all its neighbors who augment this with their local homographies to generate their own homography-to-root. This augmentation-propagation continues until all nodes have computed their homography-to-root. Note that just as in centralized homography tree technique, errors can accumulate along the paths from the root creating larger errors at PPPs which are further away from the root. In our system, we experience a maximum error of 2-3 pixels. However note that due to limitation in human perception, this error is only visible in special patterns like grids or checkerboards.

3.4 Advanced Features

Of the five capabilities mentioned in Section 2.2, till now we have achieved $a - c$. Now, we present advanced capabilities $d - e$ to realize truly scalable reconfigurable displays.

3.4.1 Adding and Removing Projectors

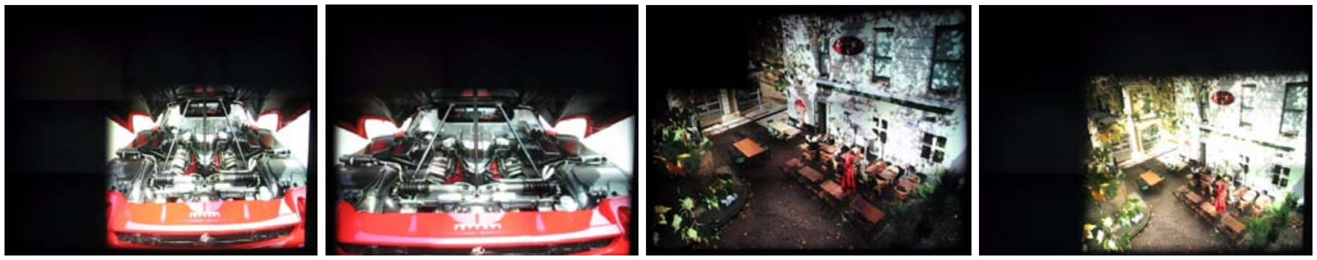
We design methods to handle addition and removal of PPPs to a calibrated display (in stable state). The cameras in the PPPs detect addition/removal of neighboring PPPs automatically and broadcast the information to all the existing PPPs. On receiving this broadcast, all PPPs then switch to the calibration phase to reconfigure the display (Pseudocode in Appendix).

Detecting Addition/Removal: Local homographies are used to segment the image into different regions, each corresponding to overlap or non-overlap region of the PPP itself or of its neighbors or empty (where no PPP projects). This segmentation can be calculated apriori during the alignment step of calibration. In the stable state, simple image processing techniques are used to detect non-black pixels in the empty region. Similarly, all black in any of the neighbor's region detects removal.

Reshaping the Display: In case of addition, a simple recalibration achieves reshaping of the display. However, since a removal creates a hole in the display, some PPPs need to be deactivated to reshape the display. For this, the message broadcasted during removal contains the coordinates of the deleted PPP. Using this information, the PPPs who are on the path to the nearest vertical and/or horizontal boundary from the removed PPP deactivate themselves. The other PPPs recalibrate to reshape the display.

3.4.2 Handling Faults

We envision hundreds of PPPs making a tiled display, especially in a public venue. In such scenarios it is a desirable



(a) (b) (c) (d)

Figure 6: Two projectors are added to a 2×2 PPP display (a) to make it a 2×3 PPP display (b). When one projector is removed from a 3×3 PPP display (c), the display reshapes itself by switching off the appropriate rows and columns to generate a 2×2 display (d).

to handle faults allowing the display to run at a lower capability even when the fault is being attended to. So, in this section, we handle the most common fault of bulb outage.

If the fault occurs in the stable state, it is handled exactly like removal of a PPP. If the fault occurs after the configuration identification step of calibration, we devise the following mechanisms to advance all the PPPs to the stable state where this is handled as a removal. Two cases occur as follows.
(a) If the faulty PPP is not the root, all the PPPs proceed to stable state automatically.
(b) If the faulty PPP is the root, the alignment stalls. We handle this using the invariant that Q must be empty after completion of Identification (since no change in patterns happen). So, the faulty root is detected by the neighboring PPPs by the existence of a non-empty Q. These PPPs broadcast a message asking everyone to advance to stable state. On receiving this message, all PPPs comply and move to the stable state.

A fault during or before the configuration identification step can be detected from the the IP-Address Table in the following manner.
(a) If the top-left PPP initiating the forward propagation fails, a *conflict* results in the IP-Address-Table with more than one PPPs having $(r, c) = (1, 1)$.
2. If any other PPP fails, it is detected as a *hole* in the IP-Address-Table i.e a possible (r, c) pair is absent.
Both the conflict and the hole can be resolved by deactivating some PPPs as in removal of PPPs in stable state. However, in this case, instead of recalibration, the other PPPs will predict the removals and update their configuration parameters (r, c, m, n) and the IP-Address-Table appropriately to instrument the reshaping in the subsequent alignment step.

3.5 Implementation and Results

We simulate a distributed asynchronous system on our 3x3 array of 9-projector. We augment each projector by a small video camera and a networked computer to simulate a PPP. For the initial algorithm design and testing we decided to work with a simulation where we simulate the distributed nature by choosing the PPP at random and the part of the SPMD code to be executed on the chosen PPP at a random granularity. This process repeats till all PPPs complete the execution of their SPMD program. Our calibration takes less than a minute assuming all PPPs to have comparable speeds. Fig 6 demonstrates addition/removal of projectors. Our video shows live results of the entire system at work. We highly encourage the reviewers to check out the following link for a high quality version of the submitted video (http://www.ics.uci.edu/~psinha/visvideo.htm). In most of the video, we deliberately slow down the calibration process by a factor of six to eight to aid better understanding.

The current implementation of our method can be made more robust. For example, the current prototype is unable to detect slight movements in the PPPs to trigger a recalibration. It can also be confused by spurious lighting in the display surroundings detecting it as addition of a PPP. Further, a black frame in one of the PPPs can be detected as removal by the adjacent PPP. However, these can be easily improved by a advanced image processing, simple network-based communications or inexpensive augmented hardware. For example, looking for the neighbor discovery patterns in the newly lighted display surroundings can easily differentiate the addition of a PPP from spurious lightings. The removal detection can be made robust by communication and processing "i am alive" messages across PPPs. This is a common approach to detect faulty nodes in distributed systems. Similarly, slight movements in the PPPs can be handled if each PPP is augmented by an inexpensive motion sensor. Note that using more than one of different types of sensors while generating such a smart device has been common in previous works. Examples include a tilt sensor in the iLamps [26] and more than one cameras in the projector bricks of [5]. Augmentation of PPPs by such devices does not reduce the effectiveness of our framework and prototype in any way. Our PPP is the minimal intelligent display unit that can provide the hot plug-and-play feature.

4 Conclusion

There has been a plethora of work on automatic calibration of multi-projector displays in the last decade. Yet, such displays are still not commonplace, the biggest inhibition being the complexity of setting them up. Our asynchronous distributed calibration methodology has the potential to remove this final barrier and make multi-projector displays truly commodity products. To the best of our knowledge, this is the first work for distributed calibration.

Some areas we would like explore in immediate future are as follows.
(a) Homography tree technique calibrates every PPP to a root PPP and is prone to inaccuracies. [3] refers to a centralized error diffusion method to address this method that is not amenable for distributed methodologies. We are currently working on local geometric calibration methodologies that can work without relying on a root PPP.
(b)We would like to address the photometric non-uniformity within and across projectors beyond blending the overlap region by designing distributed versions of the existing more rigorous photometric calibration methods like [20]. This would involve addressing the varying dynamic range and color gamut of the sensors.
(c) We would like to design an embedded hardware that can efficiently implement our distributed asynchronous method

in each PPP.
(d) We would also like to extend the method to existing display infra-structures which may not have a camera for every projector, by addressing the more general problem of calibrating m projectors using and n cameras, where $m \neq n$, in a distributed fashion.

Moving Towards Ubiquitous Pixels: Distributed calibration can have a bigger impact than just for scalable displays. *Ubiquitous pixels* - pixels anywhere and everywhere - have been envisioned by contemporary researchers as a critical component of any future workspace [7, 14]. Other critical components of future workspaces like large scale data generation and processing, ubiquitous computing, high performance networking, rendering and resource management middleware has seen significant work supported by national initiatives like TeraGrid and OptIPuter [16, 15, 9]. However, ubiquitous pixels are yet to be realized by today's display technology. The key challenges are to develop methodologies to handle non-planar, non-Lambertian and non-white surfaces. We believe that our asynchronous distributed calibration via PPPs in the first step in that direction and have tremendous potential in realizing such ubiquitous pixels "flooding" our workspaces.

References

[1] L. Childers amd T. Disz, R. Olson, M.E. Papka, R. Stevens, and T. Udeshi. Access grid: Immersive group-to-group collaborative visualization. *Proceedings of Immersive Projection Technology, Ames, Iowa*, 2000.

[2] Mark Ashdown, Matthew Flagg, Rahul Sukthankar, and James M. Rehg. A flexible projector-camera system for multi-planar displays. *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2004.

[3] Han Chen, Rahul Sukthankar, Grant Wallace, and Kai Li. Scalable alignment of large-format multi-projector displays using camera homography trees. *Proceedings of IEEE Visualization*, 2002.

[4] Yuqun Chen, Douglas W. Clark, Adam Finkelstein, Timothy Housel, and Kai Li. Automatic alignment of high-resolution multi-projector displays using an un-calibrated camera. *Proceedings of IEEE Visualization*, 2000.

[5] D. Cotting, R. Ziegler, M. Gross, and H. Fuchs. Adaptive instant displays: Continuously calibrated projections using per-pixel light control. *Proc. of Eurographics*, pages 705–714, 2005.

[6] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A.Defanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proceedings of ACM Siggraph*, 1993.

[7] T.L. Disz, M.E. Papka, and R. Stevens. Ubiworld: An environment integrating virtual reality. *Heterogeneous Computing Workshop, Geneva, Switzerland*, 1997.

[8] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[10] Mark Hereld, Ivan Judson, and Rick Stevens. Introduction to building projection-based tiled display systems. *IEEE Computer Graphics and Applications*, 2000.

[11] Mark Hereld, Ivan R. Judson, and Rick Stevens. Dottytoto: A measurement engine for aligning multi-projector display systems. *Argonne National Laboratory preprint ANL/MCS-P958-0502*, 2002.

[12] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. Wiregl: A scalable graphics system for clusters. *Proceedings of ACM SIGGRAPH*, 2001.

[13] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahem, Peter Kirchner, and James Klosowski. Chromium : A stream processing framework for interactive rendering on clusters. *ACM Transactions of Graphics*, 2002.

[14] Brad Johanson, Armando Fox, and Terry Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing Magazine 1(2)*, 2002.

[15] Yang-Suk Kee, Dionysios Logothetis, Richard Huang, Henri Casanova, and Andrew A. Chien. Efficient resource description and high quality selection for virtual grids. *In Proceedings of the IEEE Conference on Cluster Computing and the Grid (CCGrid)*, 2005.

[16] G. M. Kent, J. Orcutt, L. Smarr, J. Leigh, A. Nayak, D. Kilb, L. Renambot, S. Venkataraman, T. DeFanti, Y. Fialko, P. Papadopoulos, G. Hidley, D. Hutches, and M. Brown. The optiputer: a new approach to volume visualization of large seismic datasets. *Ocean Technology Conference*, May 2004.

[17] H.W. Kuhn. The hungarian method for solving the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[18] Kai Li, Han Chen, Yuqun Chen, Douglas W. Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Allison Klein, Zhiyan Liu, Emil Praun, Rudrajit Samanta, Ben Shedd, Jaswinder Pal Singh, George Tzanetakis, and Jiannan Zheng. Early experiences and challenges in building and using a scalable display wall system. *IEEE Computer Graphics and Applications*, 20(4):671–680, 2000.

[19] Aditi Majumder, Zue He, Herman Towles, and Greg Welch. Achieving color uniformity across multi-projector displays. *Proceedings of IEEE Visualization*, 2000.

[20] Aditi Majumder and Rick Stevens. Color nonuniformity in projection-based displays: Analysis and solutions. *IEEE Transactions on Visualization and Computer Graphics*, 10(2), March–April 2003.

[21] Aditi Majumder and Rick Stevens. Perceptual photometric seamlessness in tiled projection-based displays. *ACM Transactions on Graphics*, 24(1), January 2005.

[22] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of SIAM*, 5:32–38, 1957.

[23] Claudio Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. *Proceedings of Ubiquitous Computing, Atlanta, Georgia*, 2001.

[24] R. Raskar, M.S. Brown, R. Yang, W. Chen, H. Towles, B. Seales, and H. Fuchs. Multi projector displays using camera based registration. *Proceedings of IEEE Visualization*, 1999.

[25] 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.

[26] Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. ilamps: Geometrically aware and self-configuring projectors. *ACM Transactions on Graphics*, 22(3), 2003.

[27] L. Renambot, A. Rao, R. Singh, B. Jeong, Naveen 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.

[28] Ruigang Yang, David Gotz, Justin Hensley, Herman Towles, and Michael S. Brown. Pixelflex: A reconfigurable multi-projector display system. *Proceedings of IEEE Visualization*, 2001.

[29] Ruigang Yang, Aditi Majumder, and Michael Brown. Camera based calibration techniques for seamless multi-projector displays. *IEEE Transactions on Visualization and Computer Graphics*, 11(2), March-April 2005.

Appendix

Algorithm *Neighbor-Discovery-in-Compute*

```
begin
1.  Project Pattern;
2.  $I$ = Dequeue image from non-empty $Q$;
3.  Find all clusters in $I$; (Spatial Clustering)
4.  Find color, owner and centroid of each cluster;
5.  Create global Chromatic Blob Tables (CBT)
    for red, green, blue and white;
6.  Update neighborhood information;
end
```

Algorithm *Alignment-in-Compute*

```
begin
1.  Root = FALSE;
2.  forall $neighbors \neq \phi$ do
3.      Compute Local Homography to Neighbor;
4.      Find Overlap with Neighbor and Apply Blending;
5.  endfor
6.  if (I am the center PPP) then
7.      Root = TRUE; Homogrphy-to-Root = I;
8.  else
9.      $(H, S)$ = Recieve Homography $H$ and
        sender ID $S$ from non-empty Msg-Buf;
10.     Homography-to-Root = $H\times$ Homography-to-S;
11. endif
12. Send MSG(Homography-to-Root, myID) to all neighbors;
13. Clean Up Msg-Buf to delete unused homographies;
end
```

Algorithm *Configuration-Identification-in-Compute*

```
begin
1.  global $r = c = m = n = 1$;
2.  global $r_S = c_S = m_S = n_S = isdone = FALSE$;
3.  if (I am the top-left PPP) then
4.      $r_S = c_S = TRUE$;
5.  endif;
6.  do
7.      Encode bits in Grids and Project ID Pattern;
8.      $I$ = Dequeue image from non-empty $Q$;
9.      if all blobs in $I$ present in CBTs then
10.         Detect the IDs of the neighbor;
11.         Update $r$,$c$,$m$,$n$ and the status; (Update-IDs)
12.     else // New neighbor is detected
13.         Project Neighbor Discovery Pattern;
14.         Find new grids in $I$ and add to CBTs;
15.         Update neighborhood information;
16.         Reset $(r, c, m, n)$ to 1 and status bits to $FALSE$;
17.         if (I am the top left PPP) then
18.             $r_S = c_S = TRUE$;
19.         endif;
20.     endif;
21. until $isdone$;
22. Broadcast MSG(IP,r,c);
23. for $i = 1$ to $m \times n$ do
24.     Receive Msg from non-empty Msg-Buf;
25.     Create and Update IP-Address-Table;
26. enddo;
27. Fault-Handling;
end
```

Algorithm *Update-IDs-in-Configuration-Identification*

```
begin
1.  $(r, c) =$    $(max(r(L), r(T) + 1), max(c(L) + 1, c(T))$;
2.  $r_S =$      $(T = \phi)\|r_S(L)\|r_S(T)$;
3.  $c_S =$      $(L = \phi)\|c_S(L)\|c_S(T)$;
4.  $(m, n) = (max(r, m(B), m(R)), max(c, n(B), n(R)))$;
5.  $m_S =$      $r_S\&((B = \phi)\|m_S(R)\|m_S(B))$;
6.  $n_S =$      $c_S\&((R = \phi)\|n_S(R)\|n_S(B))$;
7.  $isdone = m_S\&n_S\&r_S\&c_S$;
end
```

Algorithm *Spatial-Cluster-in-Neighbor-Discovery* (*Blob*)

Input: An array *Blob* of (x, y) coordinates of the blobs.
Output: An array *Cluster* such that if *Blob*[*i*] and *Blob*[*j*] belong to same cluster, then *Cluster*[*i*] = *Cluster*[*j*].

```
begin
1.  for i= 1 to n
2.  Cluster[i] = i;
3.      for j=1 to n
4.          d[i][j] = dist(Blob[i],Blob[j])
5.      endfor
6.  endfor
7.  threshold = 2*min(d);
8.  change = true;
9.  while change
10. change = false;
11. for i=1 to n
12.     for j= 1 to n
13.         if (d[i][j] ¡ threshold) AND (Cluster[i] ¿ Cluster[j])
14.             Cluster [i] = Cluster [j];
15.             change = true;
16.         endif
17.     endfor
18. endfor
end
```

Algorithm *Addition-Removal-Handling-for-Compute*

```
begin
1.  Recv Msg from non-empty Msg-Buf;
2.  if (ADD-Msg) then
3.      $CALIB = TRUE$;
4.  elseif (DELETE-Msg)
5.      (nr, nc) = row and column extracted from Msg;
6.      if ($r$ between $nr$ and closest vertical boundary to $nr$) or
        ($c$ between $nc$ and closest horizontal boundary to $nc$ ) then
7.          Deactivate myself;
8.      else
9.          Update $(r, c, m, n)$ to reflect the new configuration;
10.     endif
11. endif
end
```

Algorithm *Addition-Removal-Handling-for-Capture*

```
begin
1.  Process $I$ to detect add or removal;
2.  Broadcast MSG(Add/Removal, r, c);
3.  $CALIB = TRUE$;
end
```