The Expandable Display: an ad hoc grid of autonomous displays

by

James Scott MacDougall
B.Sc., University of Victoria, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

The Expandable Display: an ad hoc grid of autonomous displays

by

James Scott MacDougall
B.Sc., University of Victoria, 2009

Supervisory Committee

Dr. Brian Wyvill, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

Supervisory Committee

Dr. Brian Wyvill, Co-Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Co-Supervisor
(Department of Computer Science)

ABSTRACT

Networking multiple "smart" displays together is an affordable way of creating large high-resolution display systems. In this work I propose a new structure and data distribution paradigm for displays of this nature. I model my work on the peer-to-peer style of content distribution, as opposed to the traditional client-server model for this kind of system. In taking a peer-to-peer approach, I present a low-cost and scalable system without the inherent constraints imposed by the client-server model. I present a new class of applications specifically designed for this peer-to-peer style of display system, and provide an easy-to-use framework for developers to use in creating this type of system.

Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Tables **vi**

List of Figures **vii**

Dedication **viii**

1 Introduction **1**

1.1 Previous Work 2

1.2 Contributions 4

1.3 Thesis Organization 5

2 The Problem to be Solved **6**

2.1 Organization 7

2.1.1 Client-Server 7

2.1.2 Peer-To-Peer 7

2.2 Data Distribution 8

2.2.1 Pixels 8

2.2.2 Primitives 9

2.2.3 Control 10

2.3 The Problem 10

3 The New Approach and Solution **12**

3.1 Overview 12

3.1.1 Novel Approach 13

3.1.2 Adapted Method 13
3.1.3 Research Basis 13
3.2 Detail 14
3.2.1 Messages 14
3.2.2 Display Nodes 14
3.2.3 Organization 16
3.2.4 Applications 23

4 Results **25**
4.1 Simulation 25
4.2 Test Applications 26
4.2.1 Balls 26
4.2.2 Flock 27
4.3 Performance Metrics 27
4.3.1 Initialization 27
4.3.2 Execution 28
4.4 Analysis 29
4.4.1 Initialization 29
4.4.2 Execution 30
4.4.3 Implications 35

5 Conclusion And Future Work **37**
5.1 Conclusion 37
5.1.1 Initialization 37
5.1.2 Execution 38
5.2 Future Work 39
5.2.1 Bootstrapping 39
5.2.2 Load Balancing 40
5.2.3 Application-Specific Improvements 40
5.2.4 Application Deployment 41
5.2.5 Other Data Types 41
5.2.6 Security 42

Bibliography **43**

List of Tables

Table 2.1 A comparison of complete display solutions. 10

Table 3.1 The new approach in the context of existing solutions. 12
Table 3.2 An example display-level message. 15
Table 3.3 An example application-level message. 16
Table 3.4 An example routing table. 17
Table 3.5 Temporary entry in the established node's routing table 21
Table 3.6 Example reply to a join request. 22
Table 3.7 Established link in the joining node's routing table. 22

List of Figures

Figure 3.1 Execution on different threads. The main execution occurs on a persistent thread, message handling occurs on a persistent thread, and message sending threads are spawned as needed. . . 15
Figure 3.2 A four-connected grid of nodes, with varying bootstrap retention distance D. Nodes in the bootstrap list are shown as grey. Small D results in a dense bootstrapped node list, while large D results in a sparse list. 20
Figure 3.3 Request to join the display system. The joining node sends four requests based on its desired position. 21
Figure 3.4 Linkages between direct neighbours. This is the default behaviour for the four-connected grid of nodes. 23
Figure 3.5 Temporary indirect linkages. These indirect neighbours are linked together to maintain the connectivity of the graph. 23

Figure 4.1 Display initialization for varying node counts and bootstrap distance D. 30
Figure 4.2 Average node join time for varying node counts and bootstrap distance D. 31
Figure 4.3 Ball application bandwidth for varying node counts and application primitives P. 31
Figure 4.4 Flock application bandwidth for varying node counts and application primitives P. 32
Figure 4.5 Average ball application update time for varying node counts and application primitives P. 33
Figure 4.6 Average flock application update time for varying node counts and application primitives P. 34

DEDICATION

This thesis is dedicated to my mother and father, Barbara and Gordon MacDougall, for their unconditional support.

Chapter 1

Introduction

Since the commodification of data projectors, networks of "smart" projectors have become a viable solution to large-format configurable display needs. Existing methods for organizing and using these displays tend to rely on the classic client-server model of data distribution. This model imposes certain constraints on these display systems that inherently limit their size and complexity.

Existing designs for composable networked displays are configurable and cost effective, but their size and complexity constraints limit their applications. Currently, displays of this nature are limited to having nodes on the order of tens, and overall display resolution dependent on server and network capabilities. Certain content distribution methods have partially addressed these matters by distributing primitives instead of pixels, offloading rendering and display to each individual node.

My proposed approach utilizes a peer-to-peer content distribution scheme, as opposed to the traditional client-server scheme. In this work each node consists of a display and a processor capable of running application code. In taking this approach, nodes are responsible for application execution, rendering, and display. This will take computational strain off a single computer (the server in previous methods), and reduce strain on the network caused by sending full pixel buffers.

My approach is built upon a display framework that supports addition and removal of nodes, and message passing between nodes. Nodes are aware only of themselves and their immediate surroundings. Special graphics applications are distributed and run across all nodes in the system. As primitive objects in these applications move through the system, they are sent from neighbour-to-neighbour, propagating through the system via the message passing system. When all display nodes are viewed in an organized manner, dependent on the physical placement of the nodes, meaningful

visual output is observed.

By using a peer-to-peer approach, larger networked displays are possible. Global constraints of previous work are traded for local constraints in this work. Resolution and scene complexity are limited only by the capabilities of individual nodes.

Whereas previous approaches only consider tens of networked displays, the end goal of this research direction is that of a much larger scale. Consider instead thousands of displays networked together to produce a massive display system.

1.1 Previous Work

An important part of display research is the use of computers to pre-modify projector output, allowing easy projection onto varying surfaces from varying angles while creating projected output that makes sense to the observer. Sukthankar et al [32] used a projector-camera pair, exploiting homography to pre-warp projector output for software keystoning. Raskar et al [29] also used a camera, and used structured light to determine surface geometry, and employed a two pass rendering method based on projective texturing to project correct imagery. Yang et al [47] took a similar approach, instead using feature points in regular projected content to determine surface geometry. Lee et al [19] used light sensors embedded in the projection surface to determine a direct projector to surface mapping.

Another important detail is correcting for color and intensity in projections. Juang [22] developed a technique for correcting luminance nonuniformity within a projector exhibited as a vignette effect. Nayar et al [38] showed how to correct for material and color of projection surfaces, allowing for reasonable visuals on varying projection surface types.

A subsequent step in display research involved taking many of these smart displays and networking them together to achieve even larger scales at a low cost. Many camera-based techniques exist for calibrating and aligning multiple displays. A popular method for determining projector-camera (and subsequently projector-projector) relationships involves detecting known projected feature points. Feature points appear in patterns that are either visible [28, 16, 20, 30, 6, 33, 31, 11], or invisible [8, 9, 21, 43, 47, 41] to the naked eye. With these relationships known, planar systems are typically corrected and aligned using planar homographies; [43, 28, 16, 30, 34, 25, 6, 33, 11]. [20] uses what is essentially a dense array of planar homographies to approximate a mapping for non-planar correction. Some systems, [31] for exam-

ple, actually estimate the display surface for use in a two-pass correction method described in [29].

With individual displays corrected and aligned, a physical framework is provided for large-scale display. Graphical information must then be generated and distributed throughout the framework to achieve meaningful output. Chapter 2 provides a survey of existing distribution techniques and networked display designs in the context of the limitations associated with each of them.

1.2 Contributions

The main contribution of this research is the design and evaluation of a display system based on a peer-to-peer content distribution model, as opposed to the usual client-server model. Instead of handling all simulation and rendering on a single node, the work is split up across all nodes by requiring nodes to handle these tasks for their own content. Furthermore, the proposed system scales well: the bandwidth and time requirements during initialization scale with the size of the display, and the bandwidth and execution time scale with the complexity of the content to be displayed.

Some important positive consequences follow from the validation of the proposed design. These consequences comprise a significant contribution to research in the field of networked composable large-format displays.

The peer-to-peer nature of the design implies that the system may be constructed with commodity components, as there is no need for a powerful master node to distribute display data to display nodes.

During display initialization, the system's bandwidth and time requirements increase at a reasonable rate with respect to the size of the display. This implies that the display system architecture can support very large display sizes.

During execution, the system's bandwidth and node execution time increase at a reasonable rate with respect to the complexity of the content to be displayed. This implies that the display system architecture can support very complex applications.

The validation of these properties suggest that a peer-to-peer framework may be better suited for building composable displays in certain situations, where display size and application complexity are a priority. These results will improve the overall size, complexity, and performance of subsequent composable display systems, and offer a framework for developing applications for such systems.

1.3 Thesis Organization

Chapter 1 contains a summary of my contributions which will be validated by this thesis, followed by an overview of the structure of the document itself.

Chapter 2 describes the research gap that is to be filled, with its context and motivation.

Chapter 3 gives the research, the new solution, and the new work done.

Chapter 4 includes the evaluation of the above solution.

Chapter 5 contains a restatement of the claims and results of the thesis. Future work for further development of the solution and its applications is discussed here.

Chapter 2

The Problem to be Solved

Creating large-format displays by joining smaller less expensive displays together has long been an area of interest in display research [29]. Central to this research is the organization of display nodes through an overlay network built upon an existing infrastructure such as TCP/IP over LAN [13, 12, 23, 37, 6, 14, 44, 30] or WAN [37, 7, 24]. With an organized network of display nodes, content may be distributed across the network for rendering and/or display.

Somewhere in the system, this organizational information must be maintained for routing purposes. Networked display research to this point has focused primarily on maintaining a client-server structure for display system networks, wherein a single source maintains organizational information of all display nodes in the system [6, 24, 33, 12, 14]. Typically, the entity responsible for maintaining this routing information is also responsible for generating and/or distributing the content needed to produce the display output for the entire system [13, 33, 12].

While existing approaches are well designed and functional, scale was never their primary goal. Design decisions made in favour of certain functionality or performance can have a negative impact on scalability. A big advantage of networking many displays together, after all, is the ability to create displays much larger than those typically available to consumers. Looking at these systems and their design decisions from a scale-centric point of view provides motivation for an alternative approach.

2.1 Organization

Given some underlying network infrastructure, applications and services require additional information and control to achieve their goals. Whether the goal is to provide unique data to individual consumers, common data to multiple consumers, or exchange data with peers, some additional information is required to organize senders and recipients.

2.1.1 Client-Server

The most common form of distribution network is that of the client-server model. This model relies on a single source for distributing data to its consumers. Web-servers follow this model, receiving and handling asynchronous requests for resources. The domain name system (DNS) translates human-readable addresses to internet protocol (IP) addresses with which resources are requested by sending messages via the internet's IP layer.

In the case of a standard web server, data is distributed via temporary connections as responses to asynchronous requests. In this situation, clients are pulling data from the server at their own pace. In the case of a synchronous service such as video streaming, which applies to some networked display systems, connections to each client must persist as data is constantly being transmitted. This imposes a fundamental limit on the number of clients served at a given moment. This limit is large (the Ajax Push Engine [1] for instance supports up to 100,000 concurrent connections) but it is a limit nonetheless.

2.1.2 Peer-To-Peer

Peer-to-peer (P2P) networks instead connect clients directly and allow for exchange of data without the need for a centralized infrastructure. With all clients acting as equally privileged agents, P2P networks exploit the sharing of resources such as bandwidth and computation. Higher overall bandwidth is achieved over models that place all bandwidth requirements on a single entity.

An example of P2P sharing in a content distribution model is multicast (network layer [10] or application layer [36]). Multicast is used for efficient delivery of common data to multiple recipients, such as streaming video to many end users [46]. Rather than maintaining an end-to-end connection between the server and each client, mul-

ticast allows a single tree-like stream of data to be sent to multiple destinations. The transmitted data follows a path common to all recipients, splitting when the path diverges. In the case of IP multicast this occurs at the actual network switches. In the case of application layer multicast, this action is carried out by the recipient clients, utilizing their upload bandwidth to contribute to the distribution. While multicast would allow for the efficient dissemination of a common piece of data it isn't exactly useful for the purposes of data distribution in multi-display systems: Each recipient node is expecting a unique piece of data, as each node displays a different part of the overall display output. The basic idea of exploiting the transmission capabilities of each node, however, is useful for such an application.

If instead the display content is generated locally at each node, this data can be passed between individual nodes as required. By distributing data in this P2P fashion, bandwidth requirements are spread across all nodes depending on the need for the data they provide. Each node would receive data from various sources in the network similar to downloading multiple files from Napster or Gnutella hosts [40], or parts of a file over BitTorrent [27].

2.2 Data Distribution

With network nodes organized and able to communicate, data may now be exchanged. In the case of real time display systems, this data falls into three categories: pixels, primitives, and control events. [15] surveyed the various content distribution methods for high-resolution multi-displays.

2.2.1 Pixels

One of the simplest ways to distribute information for display output is to directly send the framebuffer to be displayed. This method has the highest bandwidth penalty of the three, requiring large pixel buffers to be sent constantly to recipients. Some bandwidth savings may be enjoyed by using video compression techniques. It should be noted that this technique is the only option for video streaming since the source starts as pixel information. For rendered graphics, this technique is not the only option.

H.264 over IP [45] is a good example of sharing pixel data over a network in real time. Virtual networked computing (VNC) [42] is another example of sharing

pixel data, though VNC is optimized for sending changes in small subrectangles of an image, and not for full screen changes as seen in full motion video. Many networked multi-display systems are based on sending pixel data to individual display nodes, such as [33, 24, 30, 37].

When dealing with real time rendered graphics, it is important to note that pixel distribution occurs after rasterization. A graphics scene will typically consist of application primitives defined by various attributes, along with a triangle mesh and associated textures. In order to generate the pixel data to be sent, the sender must first update the application primitives through a simulation step, and render the graphics primitives, rasterizing the scene. This set of tasks imposes constraints on the logical complexity of the scene based on the sender's compute power required during the simulation step, and constraints on the graphical complexity based on the sender's graphics capabilities during rendering.

Assuming a 4K (3840 x 2160 ultra high definition television) image as a reasonable limit on the framebuffer size the server can generate, there is only enough pixel data to feed 27 display nodes at VGA (640 x 480) resolution (equation 2.1) which is not even close to the native resolution of modern consumer displays. This means that given a maximum 4K resolution for content generation from a single source, that source can only provide pixel data for 27 displays, at underwhelming VGA resolution. A limit of 27 low-resolution displays is not sufficient for a scalable multi-display system.

$$(3840 \times 2160) \div (640 \times 480) = 27 \tag{2.1}$$

2.2.2 Primitives

Another approach is to partition and send graphics primitives to recipient display nodes. In this case, display nodes are responsible for rendering the final output, and the source is responsible for the simulation step and determining which graphics primitives go to which nodes. WireGL and Chromium [12, 14] are good examples of systems that use this method. Graphics commands are sent over the network, specifying what needs to be rendered by each display node. This method significantly reduces the bandwidth requirements compared to simply sending parts of the final rasterized scene. Logical complexity of the overall scene is still constrained in this situation, as the source's computational capabilities dictate the complexity of the scene that must be updated during execution. This method of distributing primitives

is a good way to achieve scalability in terms of graphics, but logical complexity of the scene will not scale along with it.

2.2.3 Control

The final approach that consumes the least bandwidth relies on sending only user input to each node. Each display node must run a full copy of the graphics application, and it must be deterministic. Each node updates the application using the user input commands broadcast from the server. Nodes render the relevant portions of the application for their bounding areas. While simple to implement, this method imposes complexity constraints based on the capabilities of each node, requires that applications be completely deterministic, and requires a global broadcast of user input commands. PixelFlex2's PxFxLib [6] is based on this method.

2.3 The Problem

To date, scalable display systems have relied on the client-server model for organization, and one of the three data distribution schemes discussed above. Each style of data distribution has its benefits, but also imposes its own set of constraints on the content to be displayed.

Out of all the existing well-known complete systems of this nature, each has its contributions to the solution, but falls short in at least one of the following categories, in terms of scalability: node organization, data generation, data distribution, and rendering (Table 2.1).

Table 2.1: A comparison of complete display solutions.

Name	Node Organization	Data Type	Data Generation	Data Distribution	Rendering
PixelFlex [33]	centralized	pixels	centralized	centralized	centralized
PixelFlex 2 [6]	centralized	primitives / control	centralized	centralized	centralized
SAGE [24]	centralized	pixels / primitives	distributed	distributed	distributed
WireGL [12]	centralized	primitives	centralized	centralized	distributed
iLamps [30]	distributed	pixels	centralized	centralized	centralized

Pixelflex [33] and PixelFlex2 [6] use a single camera for geometric registration

to a single entity, with a single-source client-server model for content distribution. PixelFlex sends pixel data as described in section 2.2.1, while its successor supports both primitives described in section 2.2.2 and synchronized control described in 2.2.3. Because the PixelFlex systems use a single camera for geometric registration, the size of the displays are limited by the area viewed by the camera, and node organization is centralized. Furthermore, while the systems together support each of the three described distribution methods, the distributed data is still derived from a single instance of an application. This means that regardless of the display's scalability, the supported applications do not scale in a similar fashion. Scalability for both the display and its content are important aspects of the thesis research.

SAGE's [24] content distribution scheme when using the SAGE Application Interface Library (SAIL) operates more like a peer-to-peer network than other solutions. Applications run and render on one or more nodes in the system, and output is distributed to one or more nodes for display. Multiple concurrent applications may run on the system in this way. This execution and distribution paradigm is closer to the aims of this research, as multiple instances of applications can run on different nodes in the system. However, running multiple instances of applications is not the same as running a single complex application.

WireGL [12] and Chromium [14] distribute graphics-level information, which was covered in section 2.2.2. This allows for a high degree of scalability in terms of graphical complexity, but the logical complexity of the application (number of application primitives, complexity of the integration step in a physical simulation) is still limited by the source machine that is running it.

The iLamps system [30] provides a very nice registration method to allow for ad-hoc addition and removal of display nodes. This ad-hoc nature is very important for the goals of this thesis, however, the iLamps system operates by sharing pixel data from a central source which imposes constraints as described in section 2.2.1

In order to make networked multi-display systems scale better, these constraints must be relaxed or eliminated. I will present a new display organization and accompanying content generation and distribution scheme that will address the above limitations of existing display models.

Chapter 3

The New Approach and Solution

3.1 Overview

My proposed multi-display design is based on the peer-to-peer model for node organization, rather than the client-server model. Instead of relying on a single entity or server to generate and distribute display content to the rest of the system, each autonomous display node generates and displays its own content locally. In order to form a coherent scene and allow content to propagate through the system, a display primitive content distribution scheme is adopted. Nodes send high-level representations of objects to neighbouring nodes, as opposed to graphics-level primitives as seen in [12]. These high-level representations provide information which allows nodes to render and display objects locally.

By distributing every aspect of the system across display nodes, the existing research gap is filled, creating a truly scalable display system (Table 3.1).

Table 3.1: The new approach in the context of existing solutions.

Name	Node Organization	Data Type	Data Generation	Data Distribution	Rendering
PixelFlex	centralized	pixels	centralized	centralized	centralized
PixelFlex 2	centralized	primitives / control	centralized	centralized	centralized
SAGE	centralized	pixels / primitives	distributed	distributed	distributed
WireGL	centralized	primitives	centralized	centralized	distributed
iLamps	distributed	pixels	centralized	centralized	centralized
New	**distributed**	**primitives**	**distributed**	**distributed**	**distributed**

3.1.1 Novel Approach

At the heart of this system design lies the logical organization of display nodes. Since neighbouring display nodes must exchange information directly on a regular basis, the typical depth-one client-server tree is not optimal. In my system, nodes are responsible for maintaining their own logical connections to neighbouring nodes. The result is a four-connected grid of autonomous nodes with no central server infrastructure. Messages are routed on a hop-by-hop basis by each node in the path, using a coordinate routing system. While this type of communication has been used in networking [39], this is a novel approach for networked display architecture. The four-connected structure combined with coordinate routing allows the display's total coordinate space to be expanded in an ad-hoc manner through the addition of display nodes.

3.1.2 Adapted Method

Built upon this underlying peer-to-peer architecture is a system that runs applications across all nodes to produce coherent visual output. Applications are written in a parallel fashion, containing logic for simulating and rendering scenes on a single node. This logic includes conditions for sending information to, and handling incoming messages from, other nodes. When all nodes in the system run these applications in parallel, a coherent visual result emerges through local node rendering and scene objects propagating via message passing. This model is built upon a mix of regular client-side simulation and rendering, and passing of scene primitives between nodes.

3.1.3 Research Basis

This system is an alternate take on existing research on combining multiple smart displays to produce a single large display. In general, this work is a continuation on the idea that autonomous display nodes can be joined by a network interface to produce an organized controllable display system. Specifically, the initial geometric registration between new and existing nodes is built upon existing scalable display research [30].

3.2 Detail

The system is implemented in Google's Go programming language [18], and is targeted for UNIX-based systems. Testing and development is done using Mac OS X and Arch Linux operating systems [17, 2]. The application binaries depend on SDL [5] for user interface, and OpenGL [4] for graphics.

3.2.1 Messages

Display system components communicate using messages. Messages are comprised of a list of field-value pairs (Table 3.2). This list is of variable length, and must contain a specific set of field-value pairs to be considered valid. Messages are encoded as a mapping of strings to strings, for easy transmission of variable length messages with fields of heterogeneous type, marshalled into JavaScript Object Notation [3].

3.2.2 Display Nodes

The primary working component of the display system is the autonomous display node. A node has two main responsibilities; to maintain its organization within the display system, and to run application code. Each node must have a network address at which other nodes may reach it, and an associated bounding rectangle within the display system for which it is responsible. Nodes communicate with one another by sending messages described in the previous section.

Display nodes use TCP communication to send and receive information to and from other nodes. At the start of execution, each node begins listening for incoming connections on a concurrent thread. This thread in the node's client waits for incoming messages and dispatches these messages to the node's message handling function before returning to its listening state. The message handling function contains logic for calling specific functions based on the field-value pairs of messages.

When a node needs to send a message, it opens a fresh TCP connection on a new thread. This thread may either fire-and-forget a message, or send a message while blocking and waiting for a critical response (Figure 3.1). Critical responses are sent using the existing TCP connection used to transmit the original message.

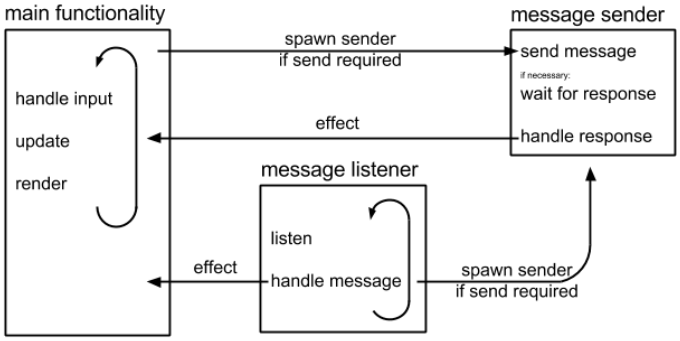


Figure 3.1: Execution on different threads. The main execution occurs on a persistent thread, message handling occurs on a persistent thread, and message sending threads are spawned as needed.

3.2.2.1 Display Layer

The display layer encompasses all behaviour related to the organization of nodes within the display system. This includes nodes joining or leaving the system, creating or destroying connections with other nodes, or modifying their bounds within the display. Messages of this sort, called display-level messages, are handled by the display layer. Table 3.2 shows an example message sent by a node wishing to join the display system at a given coordinate.

Table 3.2: An example display-level message.

Field	Value
LastAddr	142.104.69.1:9000
SenderAddr	142.104.69.1:9000
Target	Node
TargetDest	{“X”:0.0,“Y”:0.0}
Type	JoinDisplay

3.2.2.2 Application Layer

While a node uses additional threads for the network layer, its main thread of execution is responsible for simulating and rendering its application layer. The main thread spins through a loop that handles user input, performs a simulation step, and renders the application. For the time being, user input consists of keyboard and mouse events captured by the SDL library.

For one simulation step of the application, object locations and attributes are updated according to the elapsed time between the current simulation step and the previous one. This step is also responsible for updating any new objects that have arrived via messages from other nodes. Depending on the application's specific logic, it may be necessary to remove objects from the node's application and send them to another node, allowing objects to propagate through the system. In order to do this, the application constructs an application-level message that contains fields and values associated with the scene object to be sent (Table 3.3, for example, shows an application-level message). This message is then sent as described in Section 3.2.2.

Table 3.3: An example application-level message.

Field	Value
Action	AddBoid
Boid	{"Id":904643018,"Pos":{"X":45.89,"Y":51.28} ...
LastAddr	142.104.69.1:9000
LastBounds	{"Min":{"X":35.82,"Y":44.02},"Max":{"X":45.82,"Y":54.02}}
SenderAddr	142.104.69.1:9000
SenderBounds	{"Min":{"X":35.82,"Y":44.02},"Max":{"X":45.82,"Y":54.02}}
Target	Node
TargetDest	{"X":45.89,"Y":51.28}
Type	AppMessage

After simulation, the application is rendered. The objects in the node's graphics application are rendered and displayed. These objects are meant to exist within the node's logical bounds in the system. This local scene is displayed, and based on its physical position in the display system, contributes to the large overall scene.

The code for a node's execution loop remains the same, regardless of the type of application. To achieve this applications must satisfy an interface, allowing different application types to be used interchangeably.

3.2.3 Organization

Without the ability to organize themselves and communicate arbitrarily, display nodes are nothing more than singular graphics clients. In order to combine many individual display nodes, the system must establish and maintain connections between nodes.

The system design calls for a four-connected grid of display nodes. Each node only knows about, at most, four other nodes; its left, right, top, and bottom neighbours.

Coordinate-based message routing allows nodes to send messages to arbitrary nodes in the system without explicitly providing an address.

3.2.3.1 Message Routing

Each display system message has an associated two-dimensional destination point associated with it. These messages are passed node-to-node through the-four-connected grid until the message reaches its destination, where the destination is passed to the target node's message handler. Any node in the system should be able to reach any other node.

To facilitate the system's node-to-node routing, each node maintains a local routing table. Contained in this local routing table is information about each of the node's neighbours (Table 3.4). Each neighbour entry in the table specifies the node's network address and its display bounds.

Table 3.4: An example routing table.

Address	Bounds
142.104.69.1:9000	[(-10,0),(0,10)]
142.104.69.2:9000	[(10,0),(20,10)]
142.104.69.4:9000	[(0,10),(10,20)]
142.104.69.3:9000	[(0,-10),(10,0)]

Messages are routed through the system from node to node using a coordinate-based routing system similar to [39]. The structure of this system differs in that the display system deals with an unconstrained coordinate space that doesn't always have a path that resolves by repeatedly routing to the closest node. Because of this, the display system uses a second routing technique when greedy forwarding fails.

Greedy Forwarding When a node receives a message whose destination does not match its own bounds, the node must select a suitable neighbour to which it will forward the message. If there exists a neighbour in the node's routing table with bounds that lie closer to, or contain, the message destination, the message is forwarded to that neighbour (Algorithm 1). If no such neighbour exists in the node's routing table, an expanding search must be conducted to find a suitable node to which to forward. Distance here refers to Manhattan distance, an analogue to the travel between four-connected nodes similar to city blocks.

Algorithm 1: Greedy message forwarding.

```
dist = inf;
closest = nil;
for each neighbour n of forwarding node do
    if n contains destination then
        forward message to n;
        return;
    else if distance between n and destination less than dist then
        dist = distance between n and destination;
        closest = n;
if closest != nil then
    forward message to closest;
else
    discard message;
return;
```

Expanding Search If no closer neighbour exists at the greedy forwarding step, the next closest node to the destination point must be located in the system. Once this node is located, the less costly greedy forwarding may resume.

The forwarding node begins a breadth-first search for the next closest node. The search proceeds as shown in Algorithm 2, starting with the forwarding node and its neighbours comprising the search set:

Algorithm 2: Expanding node search.

```
searchset = fowarding node + neighbours;
newtoset = neighbours of searchset - searchset;
while newtoset not empty do
    for n in newtoset do
        if n closer to destination than forwarding node then
            forward message to n;
            return;
    searchset = searchset + newtoset;
    newtoset = neighbours of searchset - searchset;
discard message;
return;
```

Following these steps, a search will expand outward from the forwarding node,

resulting in forwarding the message to a suitable node if one is found, or discarding the message because its destination is unreachable. This can be costly search, but should require very few search steps before finding the next closest node in most setups. Note that nodes are searched only once, when they are first added to the search set, precluding the possibility of an infinite loop of searching.

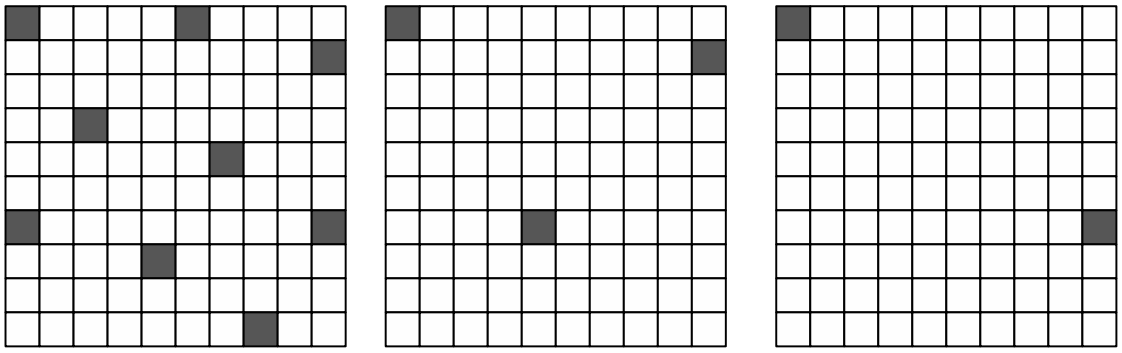
3.2.3.2 Bootstrapping

As an important detail of this messaging scheme, a sending node must know of at least one other node in the system to which it can forward a message. In the case that a new node wishes to join the display system, the joining node does not yet have any established connections to existing nodes in the system. To allow outside entry into the system, a bootstrap server provides an interface to otherwise unreachable display nodes.

The display system bootstrap server runs on an address and port that is known globally, meaning any joining node is able to communicate with it. The server's primary purpose is to receive and respond to node request messages. Node request messages specify a point in the display system's coordinate space, and expect a reply from the bootstrap server providing the address for the closest node to the requested point. The bootstrap server doesn't know about every node in the display system, as this would result in a global picture and precisely one of the scalability constraints discussed in the previous chapter. Instead, the bootstrap server maintains a sparse subset of all nodes in the system, and merely provides the closest known node to a point upon request. To further address scalability, the bootstrap server could be extended to a bootstrap hierarchy similar to a hierarchical DNS [26], where some top-level server points to other bootstrap servers, and so on.

For the purposes of this work, a single server, non-hierarchical bootstrap scheme is used. To determine whether a joining node should be added to the sparse subset of nodes maintained by the bootstrap server, a bootstrap retention distance is used. Again, distance here is defined in Manhattan distance, as an analogue to four-connected routing hops. When a node requests to join the display system through the bootstrap server, it is retained in the bootstrap's node list if the bootstrap list does not contain a node within the retention distance from the joining node.

Intuitively, a larger bootstrap retention distance results in a sparse node list, while a short distance results in a dense list. Figure 3.2, shows the difference in bootstrap



(a) D=50 (b) D=100 (c) D=150

Figure 3.2: A four-connected grid of nodes, with varying bootstrap retention distance D. Nodes in the bootstrap list are shown as grey. Small D results in a dense bootstrapped node list, while large D results in a sparse list.

list density for different bootstrap retention distances. Note that the number of nodes in the bootstrap list ultimately increases with the number of total nodes in the system, and recall that the system's bootstrap server maintains this list. The largest possible bootstrap distance that still gives "close enough" entry points should be chosen to avoid maintaining a dense or global picture of the system.

3.2.3.3 Joining The Display

For a new node to join the system, it must assume its position within the system's four-connected grid of nodes. This is done by occupying a specific area in the display system, and establishing logical connections with the neighbour nodes surrounding this area. Once this is accomplished, the node becomes a fully functioning part of the display system.

When a node joins the system, it first requests a specific position in the display system's coordinate space. This position will be within the joining node's bounds once they are established, and must not lie within any other node's bounds. Using the system's bootstrap server as an entry point, the joining node sends a request to the positions of each of its potential neighbours. This is done by sending messages to the left and right of the requested position by a system wide global definition for display node width, and to the top and bottom by the globally defined display node height (Figure 3.3). This request specifies the address at which the joining node may be reached, and indicates its desire to join the system at the requested position.

When an established node receives a request to join from a new node, it creates an

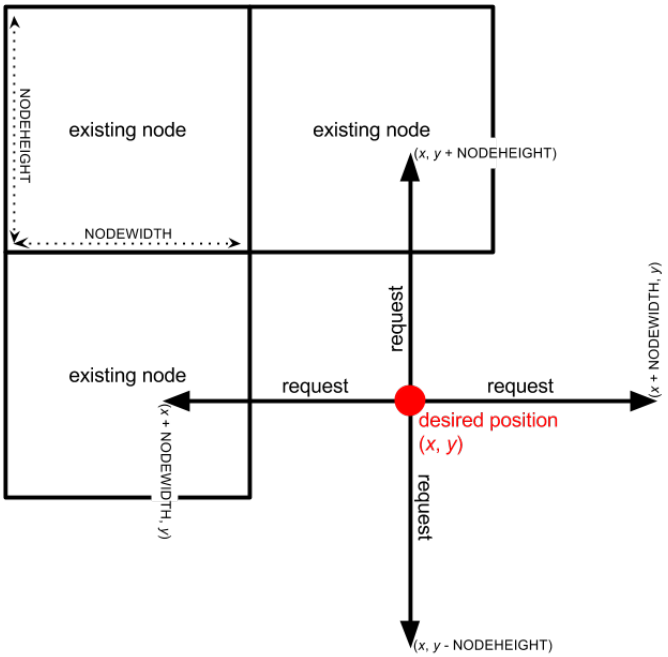


Figure 3.3: Request to join the display system. The joining node sends four requests based on its desired position.

entry in its local routing table for that node. Since the new node's bounds have not yet been established, the table entry contains the new node's address and a temporary bounds (Table 3.5). The established node then replies to the joining node with its own address and bounds (Table 3.6), allowing the joining node to create an entry for the established node in its own routing table (Table 3.7). Note that the reply message in Table 3.6 is directed to a target address rather than a target point. This is because the joining node is not yet part of the connected system and thus cannot be routed to using coordinate-based routing.

Table 3.5: Temporary entry in the established node's routing table

Address	Bounds
142.104.69.2:9000	**[(15,5),(15,5)]**

After a joining node has received replies from all of its new neighbours (either four replies are received, or a timeout occurs), it can determine its own bounds within the display system based on its neighbours. Given the bounds of its new neighbours, the joining node sets its bounds such that all neighbour bounds abut with the joining node. In the case that the joining node lacks at least one neighbour, its bounds are

Table 3.6: Example reply to a join request.

Field	Value
LastAddr	142.104.69.1:9000
LastBounds	{“Min”:{“X”:0,“Y”:0},“Max”:{“X”:10,“Y”:10}}
SenderAddr	142.104.69.1:9000
SenderBounds	{“Min”:{“X”:0,“Y”:0},“Max”:{“X”:10,“Y”:10}}
NodeAddr	**142.104.69.1:9000**
NodeBounds	**{“Min”:{“X”:0,“Y”:0},“Max”:{“X”:10,“Y”:10}}**
Target	Node
TargetAddr	**142.104.69.2:9000**
Type	**AddToTable**

Table 3.7: Established link in the joining node's routing table.

Address	Bounds
142.104.69.1:9000	**[(0,0),(10,10)]**

extrapolated based on its existing neighbours and system-defined height and width of node bounds.

3.2.3.4 Leaving The Display

For the display system to be both robust and configurable, it should be able to handle the expected and unexpected departure of display nodes. Simply ignoring departures and taking no measures to repair connectivity of the system could prove problematic.

When a node gracefully leaves the display system, it must not leave a breakage in the connectivity of the system. Since any node in the system should be able to reach any other node in the system by the routing mechanisms discussed in 3.2.3.1, a node departure must not disconnect the graph of nodes. Similarly, when a node leaves the system unexpectedly (network problem, power loss, etc.), it also must not leave a breakage in the connectivity of the system.

In order to handle both cases, each node periodically requests and maintains a list of each of its neighbours' own neighbours. Using this information, a node may establish temporary linkages between indirect neighbours when a node leaves and would otherwise cause a break in the graph (Figures 3.4 and 3.5). This is the only exception to the four-connected requirement of the system. A node may have more than four logical neighbours if it must connect to an indirect neighbour in order to maintain the connectivity of the system.

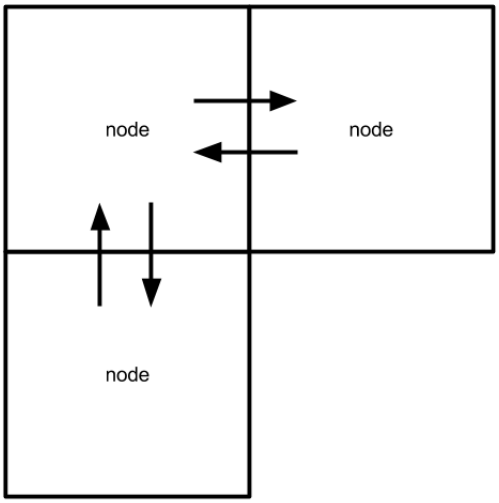


Figure 3.4: Linkages between direct neighbours. This is the default behaviour for the four-connected grid of nodes.

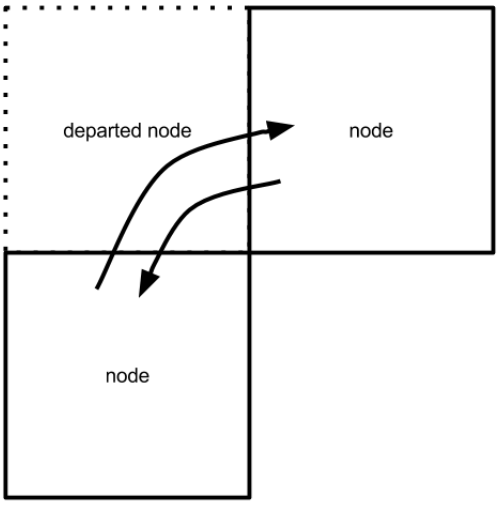


Figure 3.5: Temporary indirect linkages. These indirect neighbours are linked together to maintain the connectivity of the graph.

3.2.4 Applications

Given the parallel nature of the described system, the type of application that would be easiest to implement and enjoy the most performance would be an application that runs in parallel, segmented across all display nodes. An application of this nature is written only from the perspective of a single node, with emergent behavior coming through the execution and communication of many nodes at once.

For a simple example of such an application, consider many physical objects traveling through space, for instance in a game. The application logic would dictate the movement and collision of objects as well as rendering. As an object escapes a node's boundaries, the object is sent to the appropriate neighbour in that direction, or the object is bounced back into the bounds in the event that there is no suitable neighbour. By composing a very large number of these nodes together, a very large scene with very many objects can be achieved. Since collisions occur within node bounds, collision calculations are distributed across all nodes. Each node is responsible for rendering only its contents, so rendering is distributed as well. By scaling the number of nodes (and making some assumptions about the distribution of objects among nodes), the system can achieve a result that is far more complex and of more resolution than is possible on any one display server or monolithic display system. Note also that in this class of application, data is only communicated between nodes when a primitive needs to move from one node to the next, rather than communication occurring for every frame as in other solutions.

In the future it may be necessary to consider node-to-node visual synchronization to ensure the state of the complete scene is consistent across boundaries. Synchronization of display devices at the hardware level can be used to lock the refresh rate between two displays, while predictive sending of application primitives can overcome discontinuity imposed by network latency.

Of course, space with random objects colliding isn't a very compelling example, but it illustrates the nature of the system and its application types. More compelling examples might involve very large flocking simulations [35], or large fluid simulations where forces are communicated from node to node.

Application types that are well established in the client-server realm of multi-display systems may also be supported by this system. One could create an application that listens for and communicates with VNC servers, for instance. A server could reside outside the display system, and establish connections with display nodes via the bootstrap server. The system's coordinate-based routing may even lend itself to this kind of application, allowing the VNC server to request display on the specific nodes the content would cover based on its dimensions.

Chapter 4

Results

In this chapter I discuss data captured during display initialization and application execution, which utilize the proposed methods of the previous chapter. These captures track several performance metrics which will be used to argue the scalability of this type of system. Because large-scale testing of such a system would be costly and time consuming, I have simulated various complex display system configurations locally on a single machine. The simulation method is described below.

4.1 Simulation

The simulated system operates much like the physical system would. Nodes are still required to use the bootstrap mechanism to join the display system, and all system communications take place via the network stack. Each node operates as a separate entity, listening for and initiating display system communications on their own coroutine within the simulation application. Nodes in the simulation application use the same display system calls as the standalone node application used in real physical setups.

While the simulation application does maintain a global picture of all nodes in the system, this is only used for the rendering and control of nodes, which would otherwise be handled autonomously by individual display nodes. This global rendering and control is used only during local simulation, and in no way contributes to the design of the display system. The constraint still stands that no global picture of the display topology is maintained within the underlying system. The simulation application acts only as a graphical front end for the display system, allowing for easy addition

and removal of nodes, as well as application specific functionality such as inserting applications primitives within the display's coordinate system.

This simulation application, coupled with simple scripting for automation, allows for the execution and analysis of a wide range of display system and display application configurations. The simulation data captures several important pieces of information during execution of configurations with varying display complexity and application complexity, which will discussed in the following sections.

4.2 Test Applications

Two different applications were used during display system simulation to show how display performance may be affected by different application behaviour.

4.2.1 Balls

The first application features a set of spheres traveling at random initial velocities. As the balls reach the edge of their parent display node, they are either sent to the next node in the direction of travel, or if such a node does not exist, are bounced back into the parent node's coordinate space.

This application also features logic for displaying objects simultaneously in two or more nodes. Consider the case where a ball is situated near the edge of a display node such that its center is within the display node, but a portion of its radius extends outside the bounds of the node. In this case, the portion of the ball that extends outside its parent node should be rendered in the next node over. This is accomplished by periodically sending information about the ball to the parent node's appropriate neighbours for the purposes of rendering. The sending period can be adjusted to balance between a smooth visual effect and the bandwidth required for the sending frequency of these visual messages.

Ball locations are updated during each node's simulation step using simple Euler integration. Because ball-to-ball collision is not accounted for, this example application updates with a complexity of $O(n)$, where n is the number of balls contained in the node.

The purpose of this application is to provide data pertaining to a display application with linear complexity, with a fairly uniform distribution of application primitives across all display nodes. Display performance in a system with simulta-

neous application primitive rendering in multiple nodes is also of great importance, because without this feature the overall system visuals would be discontinuous and displeasing to the eye.

4.2.2 Flock

The second application features autonomous flocking agents, as described by Craig Reynolds [35]. Application primitives are constantly changing velocity as they flock together through the overall display system. As in the ball application, flocking agents are passed between nodes as their position and velocity dictates. The agents also feature additional logic causing them to steer away from node edges when no available neighbour node exists in the direction of travel.

Flocking agent locations are again updated using Euler integration, however here the acceleration component is derived from a $O(n^2)$ algorithm, where n is the number of flocking agents in a given node.

This application provides data related to a display application with quadratic complexity. The distribution of application primitives here is non-uniform, and may heavily load certain nodes with application primitives, depending on flocking behaviour.

4.3 Performance Metrics

During display simulation I have recorded several performance metrics to help argue the scalability of such a display system. These metrics demonstrate the system's performance as a whole by considering bandwidth requirements for certain configurations, and local performance in the form of computational performance.

4.3.1 Initialization

During system setup, nodes are added to the system and aligned with existing nodes. During this initialization step, two performance metrics are recorded: bandwidth usage and join time.

4.3.1.1 Display Setup Bandwidth

Display setup bandwidth refers to the amount of overall bandwidth required to initially set up the structure of the display system. During the setup phase the simulation application logs the amount of display-level message (as described in Section 3.2.2.1) data sent. This performance metric depends on two variables: the number of nodes in the final display configuration, and the bootstrap retention distance as described in section 3.2.3.2. To satisfy the requirement of scalability, the required bandwidth during setup should not exceed linear growth with respect to the number of nodes in the system.

4.3.1.2 Node Join Time

Node join time refers to the amount of time it takes from the point at which a node requests to join the system, to the point that node is fully connected and aligned within the system. Like display setup bandwidth, this metric depends on the number of nodes in the final display configuration, and the bootstrap retention distance used. Similarly, node join time should not exceed linear growth with respect to the number of nodes in the system.

4.3.2 Execution

Once the system is initialized, execution of a display application (Section 3.2.4) may begin. During execution, two independent performance metrics are recorded: bandwidth usage and performance.

4.3.2.1 Application Execution Bandwidth

Application execution bandwidth refers to the amount of data sent during the run of the simulation. This data is comprised of application-level messages described in Section 3.2.2.2. This bandwidth metric depends on two variables: the number of nodes in the display system, and the total number of application primitives in the system.

4.3.2.2 Application Performance

The final performance metric refers to the time it takes a display node to advance the state of its application during a single execution step. In the case of a physics-

based game, this could include Euler integration and other game-specific logic. This metric will give an indication of how certain types of applications will perform, given the complexity of the display configuration and the overall number of application primitives in the simulation. Ideally, this metric would show that an increase in the number of display nodes reduces the load on the nodes overall, meaning an increase in display complexity allows for a similar increase in application complexity without performance penalty.

4.4 Analysis

This section presents the recorded performance metrics described in the previous section, and their meaning with regards to system performance on a much larger scale. Extrapolating patterns in performance can give insight into how the system might perform at scales greater than those simulated.

4.4.1 Initialization

Here follow the results for the performance metrics described in Section 4.3.1.

4.4.1.1 Display Setup Bandwidth

Figure 4.1 shows the bandwidth requirements related to display initialization. This data shows a clear superlinear growth with respect to the number of nodes in the system. While this effect is shown to be reduced when selecting a shorter bootstrap distance, the growth rate is still clear. This is not ideal for scalability, but at the very least the data shows reasonable bandwidth requirements for very large node counts. These node counts are much larger than those seen in competing display system designs, which are typically in the range of tens rather than hundreds.

Recall from Section 3.2.3.2 that the largest possible bootstrap distance that still gives "close enough" should be chosen; in this case bandwidth requirements that can be met by the underlying network. Note that, while a smaller bootstrap distance is less demanding on bandwidth, smaller bootstrap distances lead to denser node lists and approach a global picture of the system. Maintaining a global picture of the system must be avoided to meet scalability constraints.

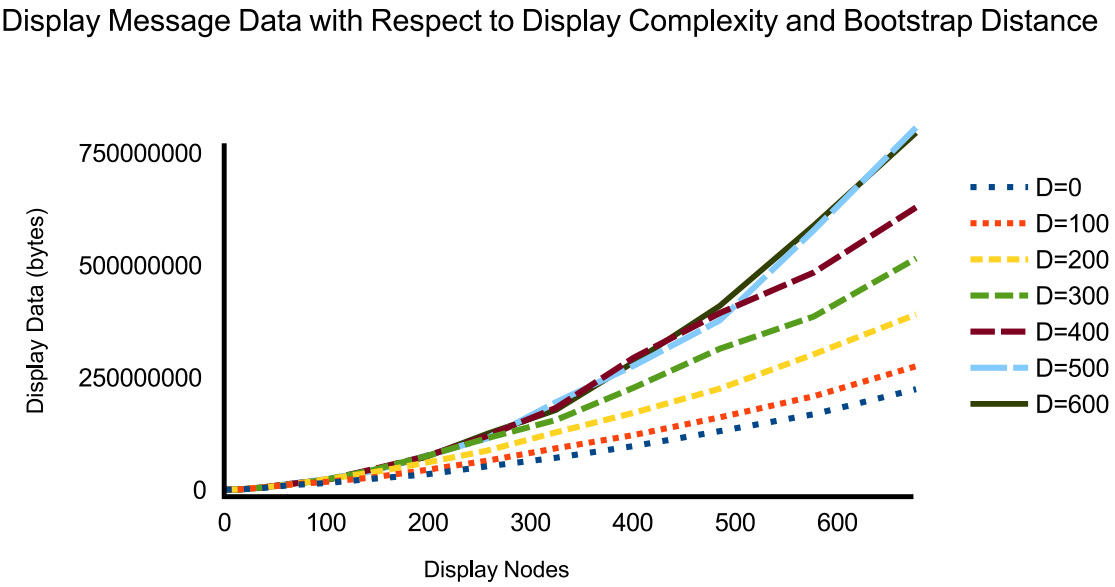


Figure 4.1: Display initialization for varying node counts and bootstrap distance D.

4.4.1.2 Node Join Time

The previous section showed that more forwarding does not increase bandwidth significantly. The same does not apply to node join time. Figure 4.2 shows that the number of nodes in the system and the bootstrap retention distance both significantly affect the average join time for display nodes. When the bootstrap retention distance is large enough, so large that the only node in the bootstrap list is the original node in the system, join time grows at a linear rate as the number of nodes increases. The shorter the bootstrap distance is, and the denser the bootstrapped node list is, the less time it takes on average for a node to join the system. This metric clearly shows the benefit of having a shorter bootstrap retention distance.

4.4.2 Execution

Here follow the results for the performance metrics described in Section 4.3.2.

4.4.2.1 Application Execution Bandwidth

In this section the scalability of the actual execution of applications is analyzed. While system initialization scales well with the number of nodes in the system, application

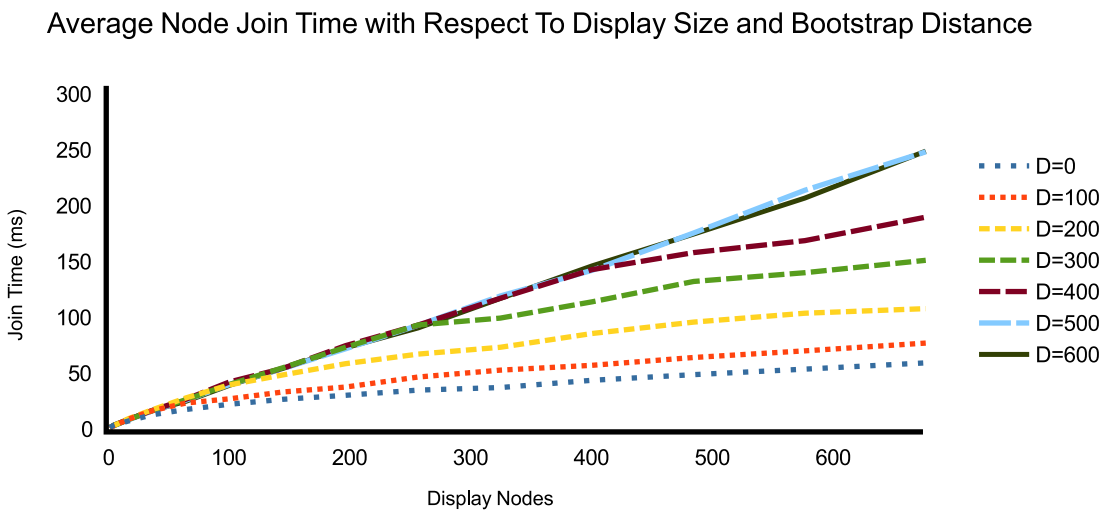


Figure 4.2: Average node join time for varying node counts and bootstrap distance D.

network activity must show at most linear growth in order for the complete system to scale well.

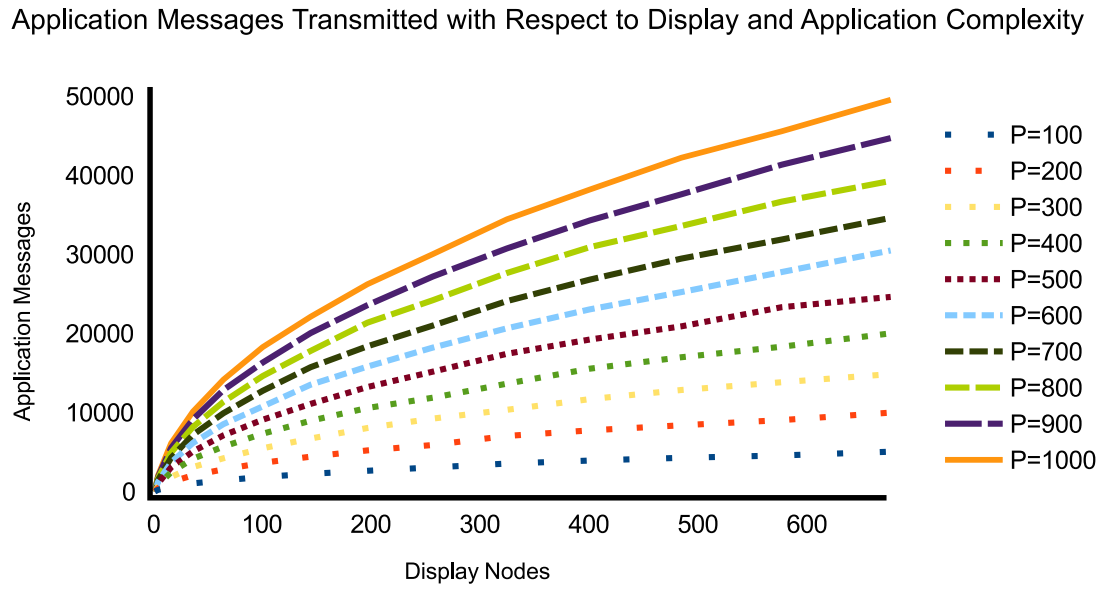


Figure 4.3: Ball application bandwidth for varying node counts and application primitives P.

Figure 4.3 shows the results for executing the ball app described in section 4.2.1.

Individual curves denote the bandwidth required for various display setups operating with different numbers of application primitives in the system. The display system is set up in each simulation to be a square grid of nodes covering the same coordinate space. All curves show a growth in network activity that is sublinear with respect to the number of nodes, which satisfies the scalability constraints of the system. For any given number of display nodes, the individual curves are uniformly spaced indicating linear growth of bandwidth requirements with respect to application complexity.

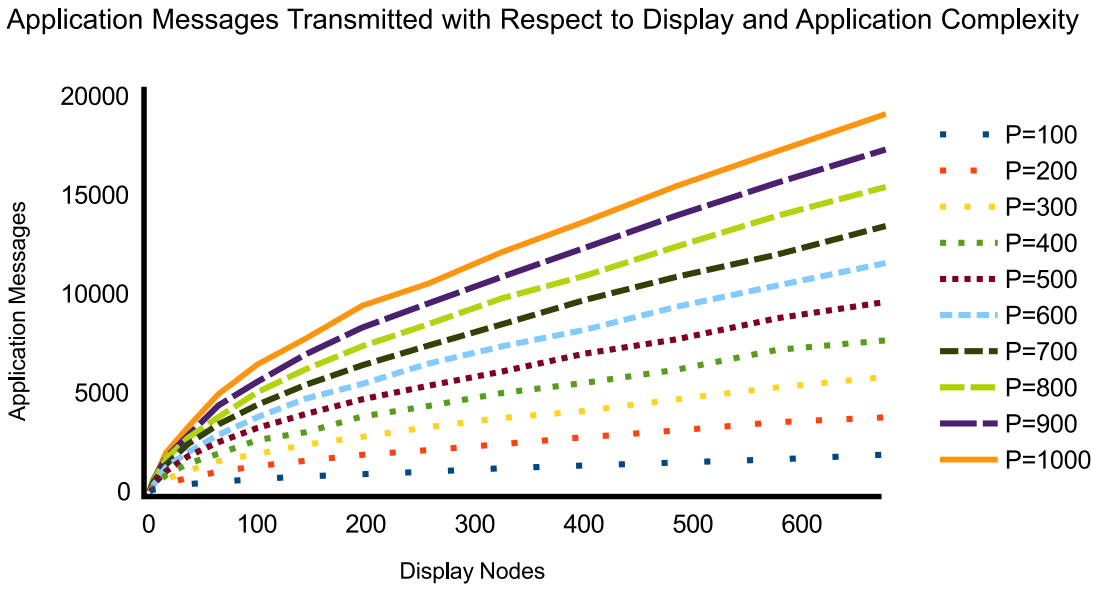


Figure 4.4: Flock application bandwidth for varying node counts and application primitives P.

Figure 4.4 shows the results for the flocking app described in section 4.2.2. As with the ball application simulations, the display is set up as a square grid, always subdividing the same coordinate space. All curves again show sublinear growth to start, with respect to the number of nodes, possibly increasing to linear growth as the number of nodes exceeds 300. Again like the ball application simulations, the flock simulations show linear growth in bandwidth with respect to application complexity.

The results from both application type indicate good scalability. Bandwidth requirements of the system scale at most linearly with both display and application complexity; bandwidth scales sublinearly (transitioning to linear in the flocking example) with display complexity, and roughly linearly with application complexity. A reasonable and logical requirement for physically building such a system would re-

quire that network infrastructure scales linearly with the complexity of the system and its intended applications.

4.4.2.2 Application Performance

The previous two sections have shown data supporting the idea that the system's communications scale well for both initialization and execution, but do not support the idea of computational scalability. Arguably the most important aspect of such a system is the final visual output; the aspect of the system that the users interact with directly.

Intuitively, increasing display nodes while keeping application primitives constant should translate to better overall performance, in the form of shorter average application update times. Adding nodes to the system allows for better distribution of application primitives across the system, allowing for better individual node performance. This behaviour should hold for relatively uniform distribution of primitives, but it is unclear what might happen in less uniform situations such as flocking.

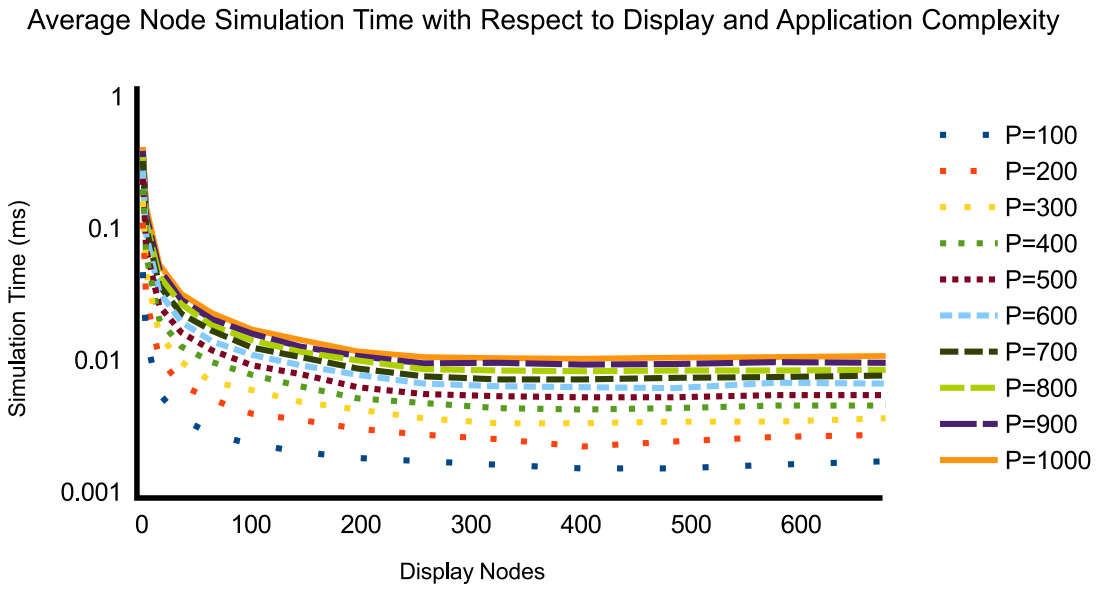


Figure 4.5: Average ball application update time for varying node counts and application primitives P.

Figure 4.5 shows the average application performance for the ball application described in 4.2.1. As intuition suggests, an increase in display nodes translates to

better overall performance for individual display nodes. Increasing the number of application primitives for a given display configuration is shown to incur a roughly linear performance cost (note the logarithmic scale on the y-axis).

Figure 4.5 showed the average performance of nodes, but what would worst-case performance look like? The worst-case for any number of nodes should never exceed the worst-case for a single-node setup; In a single-node setup, all application primitives exist in a single node and the maximum possible computational penalty is incurred, while in multi-node setups, the same number of primitives is distributed across all nodes, with the worst-case distribution resulting in all primitives residing in a single node. Due to the autonomous nature of the display system, a node in a single-node setup and a node in a multi-node setup will perform equivalently for the same application complexity.

In the case of the ball app, application primitives are always relatively uniformly distributed across all nodes. In this case the worst-case performance for a node is rarely, if ever, achieved for a multi-node setup.

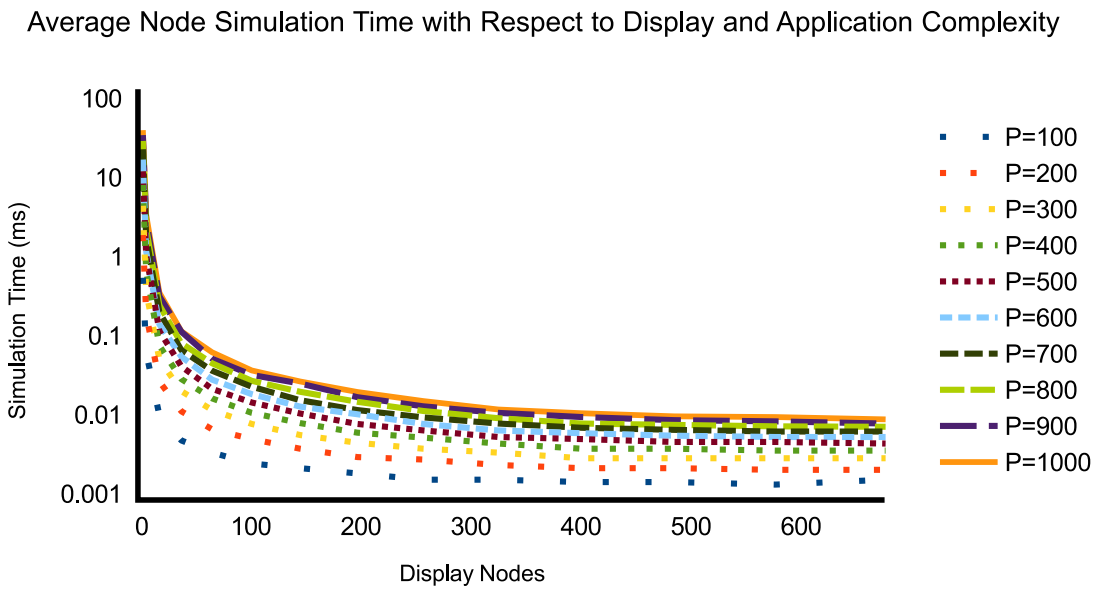


Figure 4.6: Average flock application update time for varying node counts and application primitives P.

Figure 4.6 shows performance for the flocking app described in 4.2.2. Like the ball app, the flocking app shows predictable performance savings when increasing the number of display nodes. Even though the application primitives are less uni-

formly distributed across the display nodes, the system still enjoys better performance overall.

Also shown by each of the graphs is an increase in simulation time, or decrease in performance, for a similar increase in application complexity defined by the number of application primitives in the simulation. The results for both types of applications are consistent with the performance claims laid out in Section 3.2.4.

As with the ball app, worst-case performance of a node should never exceed the worst-case performance of a single-node setup. Even though flocking primitives tend to bunch together and cause a less uniform distribution of primitives over all nodes, the system still enjoys performance increases similar to those in the ball app.

4.4.3 Implications

This section has shown evidence to support the claim that the system described in the previous chapter is scalable in most aspects. The results in section 4.4.1.1 show that display initialization, however, has bandwidth requirements that grow superlinearly with respect to the number of nodes in the display. If the display system's network infrastructure is built up at the same rate as its display nodes, the initialization aspect of the system will not scale indefinitely. This bandwidth usage though is amortized across the time it takes to physically set up the display nodes.

Section 4.4.1.2 shows that node join time increases linearly with the number of nodes in the display system, and can be kept sublinear with appropriate selection of the bootstrap retention distance. Note though this is the average join time, and not the sum total of all node join times. The total join time would show similar superlinear growth with respect to the number of nodes. Recall though that the autonomous nature of display nodes means joining nodes only affect a very localized area of the display. Due to this property, display initialization can be parallelized by adding nodes concurrently as long as their affected localized areas do not overlap.

Section 4.4.2.1 shows that for increasingly large displays covering the same coordinate space, application bandwidth requirements increase at a sublinear rate. On the other hand, application bandwidth requirements increase linearly for increasingly complex applications, where complexity is defined as the number of display primitives in the system.

Finally, section 4.4.2.2 shows that increases in display size result in a similar increase in individual node performance. Increases in application complexity show a

similar decrease in individual node performance. A sufficiently sized display system can maintain interactive speeds overall for large numbers of application primitives. More nodes means support for more application primitives without performance penalties.

Based on these results, it follows that the proposed system is a scalable one. Bandwidth requirements scale well for proportionally increasing display size and application complexity, during both initialization and execution. Computational performance also scales well when display size and application complexity increase proportionally.

Chapter 5

Conclusion And Future Work

5.1 Conclusion

In this work I designed a networked composable display system based on a peer-to-peer system architecture. By implementing a reference system I was able to validate the design. The system operates as originally described in Chapter 1, validating the claim that such a system may be constructed in a peer-to-peer fashion. As stated in Section 1.2, this implies that such a display system may be constructed of commodity components without the need for any specialized equipment.

While the initial claim is validated, it says nothing about the actual utility of such a system. Without adequate performance to back it up, the new system isn't of much use. In Section 1.2 two additional claims are laid: the bandwidth and time required during display initialization scales in proportion to the number of display nodes comprising the system, and the bandwidth and individual node performance during execution and display scale in proportion to the complexity of the content being displayed. The first claim was partially validated in Section 4.4.1, and the second claim was validated in Section 4.4.2. The system's initialization will not scale indefinitely with respect to the number of display nodes, but nodes on the order of thousands are supported. The results of these claims show that the system is a good one, but does not scale indefinitely in all aspects.

5.1.1 Initialization

Due to the ad-hoc nature of the proposed display system, a collection of autonomous nodes that may leave and join the system at their own discretion, there is not neces-

sarily a specific initialization period; nodes are added to the system as needed during its lifespan, based on the desires of the user. For the express purpose of evaluating the system, I considered an initialization period where the total number of desired display nodes are added to the system in quick succession. Following the procedure described in Sections 4.3.1.1 and 4.3.1.2, I captured data corresponding to the associated bandwidth, and time requirements respectively.

Section 4.4.1.1 showed the relevant data and made the argument that, while showing distinct quadratic growth in the worst case where the bootstrap short enough, the bandwidth requirements still show very reasonable values for display sizes greater than any seen previously with existing networked display approaches. Display initialization should be improved, namely the bootstrapping procedure, before a claim is made for true scalability, but the display system scalability still far exceeds the needs of any existing physical display.

I examined the average time required for a display node to join the system during the initialization procedure, explained in section 4.4.1.2 . Figure 4.2 showed a worst-case linear growth in time, with adequate bootstrap distance selection showing very good sublinear growth. Linear to sublinear growth shows very good scalability in terms of join time, which is arguably more important than any bandwidth requirement as it translates directly to interactivity to the user.

Based on the relevant data, the claim regarding requirements for display initialization is validated. All that remains is the validation of the similar claim regarding display application execution.

5.1.2 Execution

The real value of the connected display system shines with the generation and display of graphics. The display system has already been shown to support the creation of very large displays, and now these large displays are shown capable of efficiently generating visual output. This capability was characterized in Sections 4.3.2.1 and 4.3.2.2 as the required bandwidth for application-related communications, and the performance of individual nodes during execution.

Section 4.4.2.1 showed the required bandwidth for application-related communications during execution. The data for two different applications both showed sublinear growth in these bandwidth requirements when subdividing an application's coordinate space by using more display nodes. Based on the argument that the underlying

network should scale linearly with the size of the physical display, this shows promise for scalability of bandwidth usage.

Section 4.4.2.2 showed the average node simulation time during execution. As expected, subdividing an application's coordinate space by using more display nodes resulted in better performance on average per node. Better performance for the same application complexity implies support for very complex applications when there are sufficient display nodes in the system to take the execution load. Note that the best performance savings come when application primitives are distributed evenly across the coordinate space and subsequently the display nodes.

Based on this data, the claim that the bandwidth usage and node performance scales during execution is validated. With the two previous claims being shown as valid, the quality of the display system solution is validated.

5.2 Future Work

The display system proposed in this work has been shown to be valid, but there are areas with room for improvement. All aspects of the system have been shown to scale with at-worst linear growth when growing the size of the display or the complexity of the graphics application, except for the bandwidth required for the bootstrap procedure examined in Section 4.4.1.1. Also in need of exploration are the potential performance savings in load balancing across nodes regardless of the actual positioning of application primitives in the display coordinate space, application-specific improvements, a way to easily deploy application code to nodes in an existing system, constrained support for other data distribution methods, and security.

5.2.1 Bootstrapping

As mentioned in 4.4.1.1, the bootstrapping and alignment procedure for connecting new nodes to the existing display system grows at a superlinear rate as the number of nodes in the system increases. The likely cause of this is the amount of breadth-first forwarding of messages that must be done to exhaust the search for a node that doesn't exist at the message destination point (Figure 3.3 shows two such points). If instead the bootstrap server and coordinate-based routing of requests can be eliminated, the bandwidth requirements should drop considerably. Future work on this issue may involve using a vision-based node discovery method similar to that seen in [30]. In a

physical deployment of the display system, nodes could have projector-camera pairs. A joining node could conceivably display a pattern indicating its request to join the display, which would only be seen by nearby nodes. This pattern could have the new node's information encoded, allowing for existing nodes to begin the alignment procedure without the need for a bootstrap server or message forwarding. Such an improvement could introduce constant-time and constant-bandwidth joining for nodes.

Another option for improving bootstrap performance is support for a dynamic bootstrap retention distance. Rather than enforcing a static spacing between nodes known by the bootstrap server, a more suitable approach would be to enforce that the bootstrap server knows of some percentage of total nodes instead. In taking this approach, the user wouldn't have to predict the size of the display when choosing an initial bootstrap retention policy. BitTorrent [27] takes a similar approach to bootstrapping, restructuring its list of peers based on their performance.

5.2.2 Load Balancing

As noted in the node performance analysis, the best performance savings when subdividing an application's coordinate space with many nodes is when application primitives are uniformly distributed across all nodes. This is obviously not the case in most situations, especially when user interactivity is concerned. In the case of a simple arcade game, it is likely that many "enemy" application primitives would flock toward a user-controlled "player" primitive, precluding any uniform distribution of primitives across the display system. In this case one could imagine a small number of nodes being heavily loaded with primitives, while most other nodes in the system contain very few primitives and thus have plenty of unused computational capacity. It may be worth exploring methods for redistributing this surplus computational capacity to help the more heavily loaded nodes with their workloads. In exploring this, one should consider whether ithe performance savings are worth affecting the relative simplicity and ad-hoc nature of the existing display system design.

5.2.3 Application-Specific Improvements

For simplicity of implementation and testing, the two example applications used in this work lack the polish of a presentation-grade application. In their current form, the two applications operate under the limitation that any given application primitive

exists only in a single display node at a given time. Clearly this should not be the case if an application primitive lies very close to a node boundary, and its size is large enough to encroach on an adjacent node's coordinate space. In such a case, the node should be represented in both nodes in order to form a coherent visual representation of the primitive as it spans more than one node. Such a thing is certainly possible using the existing node communication methods on an app-by-app basis, but it would be worth exploring whether such a feature could be implemented in such a way that this behaviour becomes automatic.

5.2.4 Application Deployment

As of now, the specific graphics applications used by the display system are part of the display system code base and are pre-compiled in the display system binaries. In a real deployment of such a system, this is obviously not ideal. Ideally an application programmer should be able to create a specific graphics application, and either pre-compile or simply use the source as input to the system. The system should take this input at an entry point, say the bootstrap server, and propagate the new application through the system in a breadth-first fashion. The ability to easily distribute new applications to an existing display system would be very important in real-world applications where a display system installation would ideally exist in an always-on state.

5.2.5 Other Data Types

One clear limitation of the proposed system is that only the primitive display data type is used. Pixel sharing, or video streaming, is however very important for large format displays; full motion video playback is far and away the most popular use for large displays. Due to the single-source nature of video streaming, such a feature would not enjoy the same performance as the primitive data model, but it would still be useful to support a limited version of pixel sharing. Perhaps a client-server overlay could be created on top of the peer-to-peer node layout, allowing users to share pixel buffers to a subspace of the complete display system.

5.2.6 Security

Since the reference design was implemented and used in a controlled local environment, there was no real need for any security measures in the display system architecture. Obviously in a real deployment of the system, security would be very important. Work should be done on using encrypted communications and authentication to avoid privacy issues and block various attack vectors.

Bibliography

[1] Ajax push engine. http://ape-project.org/.

[2] Arch linux. http://www.archlinux.org/.

[3] Javascript object notation. http://www.json.org/.

[4] Opengl. http://www.opengl.org/.

[5] Simple directmedia layer. http://www.libsdl.org/.

[6] A. Majumder H. Towles A. Raij, G. Gill and H. Fuchs. Pixelflex2: A comprehensive, automatic, casually-aligned multi-projector display. In *International Workshop on Projector-Camera Systems*, pages 203–211. IEEE, 2003.

[7] R. Jagodic R. Singhm J. Aguilera A. Johnson B. Jeong, L. Renambot and J. Leigh. High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 24–24. IEEE, 2006.

[8] M. Gross D. Cotting, M. Naef and H. Fuchs. Embedding imperceptible patterns into projected images for simultaneous acquisition and display. In *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 100–109. IEEE, 2004.

[9] M. Gross D. Cotting, R. Ziegler and H. Fuchs. Adaptive instant displays: Continuously calibrated projections using per-pixel light control. In *Computer Graphics Forum*, volume 24, pages 705–714. Wiley Online Library, 2005.

[10] S.E. Deering and D.R. Cheriton. Host groups: A multicast extension to the internet protocol, 1985. RFC 966.

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

[12] I. Buck G. Stoll M. Everett G. Humphreys, M. Eldridge and P. Hanrahan. Wiregl: a scalable graphics system for clusters. In *SIGGRAPH*, volume 1, pages 129–140, 2001.

[13] M. Eldridge G. Humphreys, I. Buck and P. Hanrahan. Distributed rendering for scalable displays. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, page 30. IEEE Computer Society, 2000.

[14] R. Ng R. Frank S. Ahern P. Kirchner G. Humphreys, M. Houston and J. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 693–702. ACM, 2002.

[15] A. Finkelstein T. Funkhouser K. Li Z. Liu R. Samanta H. Chen, Y. Chen and G. Wallace. Data distribution strategies for high-resolution displays. *Computers & Graphics*, 25(5):811–818, 2001.

[16] G. Wallace H. Chen, R. Sukthankar and K. Li. Scalable alignment of large-format multi-projector displays using camera homography trees. In *Proceedings of the conference on Visualization'02*, pages 339–346. IEEE Computer Society, 2002.

[17] Apple Inc. Os x mountain lion. http://www.apple.com/osx/.

[18] Google Inc. The go programming language. http://http://golang.org/.

[19] D. Maynes-Aminzade R. Raskar J. Lee, P. Dietz and S. Hudson. Automatic projector calibration with embedded light sensors. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 123–126. ACM, 2004.

[20] S. Roy J. Tardif and M. Trudeau. Multi-projectors for arbitrary surfaces without explicit calibration nor reconstruction. In *Proceedings. Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003.*, pages 217–224. IEEE, 2003.

[21] A. Akbarzadeh J. Zhou, L. Wang and R. Yang. Multi-projector display with continuous self-calibration. In *Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems*, page 3. ACM, 2008.

[22] R. Juang and A. Majumder. Photometric self-calibration of a projector-camera system. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07*, pages 1–8. IEEE, 2007.

[23] B. Wylie K. Moreland and C. Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 85–92. IEEE Press, 2001.

[24] R. Singh-B. Jeong N. Krishnaprasad V. Vishwanath V. Chandrasekhar N. Schwarz A. Spale L. Renambot, A. Rao and C. Zhang. Sage: the scalable adaptive graphics environment. In *Proceedings of WACE*, volume 9, pages 2004–09. Citeseer, 2004.

[25] T. Okatani and K. Deguchi. Easy calibration of a multi-projector display system. *International journal of computer vision*, 85(1):1–18, 2009.

[26] J. Postel. Domain name system structure and delegation, 1994. RFC 1591.

[27] D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 367–378. ACM, 2004.

[28] J. van Baar R. Raskar and J. Chai. A low-cost projector mosaic with fast registration. In *Asian Conference on Computer Vision (ACCV)*, volume 3, 2002.

[29] M. Cutts-A. Lake L. Stesin R. Raskar, G. Welch and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 179–188. ACM, 1998.

[30] P. Beardsley T. Willwacher S. Rao R. Raskar, J. van Baar and C. Forlines. ilamps: geometrically aware and self-configuring projectors. In *ACM SIGGRAPH 2006 Courses*, page 7. ACM, 2006.

[31] R. Yang W. Chen G. Welch H. Towles B. Scales R. Raskar, M. Brown and H. Fuchs. Multi-projector displays using camera-based registration. In *Visualization'99. Proceedings*, pages 161–522. IEEE, 1999.

[32] R. Stockton R. Sukthankar and M. Mullin. Smarter presentations: Exploiting homography in camera-projector systems. In *Proceedings. Eighth IEEE International Conference on Computer Vision, 2001. ICCV 2001.*, volume 1, pages 247–253. IEEE, 2001.

[33] J. Hensley H. Towles M. Brown R. Yang, D. Gotz. Pixelflex: A reconfigurable multi-projector display system. In *Proceedings of the conference on Visualization'01*, pages 167–174. IEEE Computer Society, 2001.

[34] A. Raij and M. Pollefeys. Auto-calibration of multi-projector display walls. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 14–17. IEEE, 2004.

[35] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[36] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM 2002 Conference*. ACM, 2002.

[37] S. Daly S. Deshpande, C. Yuan and I. Sezan. A Large Ultra High Resolution Tiled Display System: Architecture, Technologies, Applications, and Tools. In *Proceedings of the 16th International Display Workshops*.

[38] M. Grossberg S. Nayar, H. Peri and P. Belhumeur. A projection system with radiometric compensation for screen imperfections. In *ICCV workshop on projector-camera systems (PROCAMS)*, volume 3. IEEE Press, 2003.

[39] M. Handley R. Karp S. Ratnasamy, P. Francis and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001 Conference*. ACM, 2001.

[40] K. Gummadi S. Saroiu and S. Gribble. Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia systems*, 9(2):170–184, 2003.

[41] H. Fuchs E. La Force T. Johnson, G. Welch and H. Towles. A distributed cooperative framework for continuous multi-projector pose estimation. In *Virtual Reality Conference, 2009. VR 2009. IEEE*, pages 35–42. IEEE, 2009.

[42] K. Wood T. Richardson, Q. Stafford-Fraser and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33–38, 1998.

[43] K. Ito T. Aoki T. Takahashi, T. Kawano and S. Kondo. Performance evaluation of a geometric correction method for multi-projector display using sift and phase-only correlation. In *ICIP*, pages 1189–1192, 2010.

[44] J. Klosowski W. Corrêa and C. Silva. Out-of-core sort-first parallel rendering for cluster-based tiled displays. *Parallel Computing*, 29(3):325–338, 2003.

[45] S. Wenger. H. 264/avc over ip. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):645–656, 2003.

[46] Y. Guo Y. Liu and C. Liang. A survey on peer-to-peer video streaming systems. *Peer-to-peer Networking and Applications*, 1(1):18–28, 2008.

[47] R. Yang and G. Welch. Automatic projector display surface estimation using every-day imagery. In *9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2001.