Visual Computing in the Future: Computer Graphics as a Remote Service

Dale Beermann, Greg Humphreys
University of Virginia
Computer Science Department
{beermann, humper}@cs.virginia.edu

June 1, 2003

1 Introduction

The focus of Visual Computing has changed quite dramatically over the past few decades. Previously our concern was with very low level algorithms but we have progressed beyond much of this to tackle new issues. Many of the problems presented in Visual Computing have dealt with the general problem of not being able to display a dataset fast enough, or even at all. The first solutions to this have been labeled with the now ubiquitous term "big iron." Supercomputers provided the only solution for a long time but they were expensive. We have now reached the point where commodity graphics hardware improvement rival what many supercomputers were designed to do. As a result, the trend is moving toward commodity based clusters as a solution to big data constraints, providing a cost-effective alternative that is challenging the market for supercomputers. Another result of the growing size of datasets is that we need to develop new displays to view them; the resolution of a typical desktop display is not adequate for viewing the extreme detail we can now incorporate into models. Because of this, display walls using tiled projectors and specialized high-resolution displays have emerged. Of course, as we continue to find solutions to problems like these, new ones arise that must be addressed.

One of the main issues we see in Visual Computing today is the tight coupling of the graphics hardware and its display. It is very difficult to view the results of a graphically intense application from a different location. Many applications require the rendering power of a cluster or supercomputer, but one isn't always available. To view an application, a specialized visualization center may be needed. The problem is that the display must be located in close vicinity to the cluster or supercomputer controlling it. Since most people won't have either of these in their office their work needs to be done elsewhere. This can present a considerable inconvenience, especially if the system isn't located in the same building. Imagine a scenario where you only want to view a minute part of a dataset, maybe for reference when writing a paper, but you don't need to use a display wall. It would be very beneficial to be able to do so at your desk. This would save the time required to go to the visualization center, boot up the cluster, turn on all the projectors, make sure they're calibrated, start the application, make your observations, and then shut everything down.

We envision a future where graphics hardware has been decoupled from any display. As a result, visual computing can take on the role of a service. A cluster located in the basement of another building, not necessarily even in the same city or country, could be accessed via a web page or some other interface. A job can be loaded, a specific number of nodes allocated to the application, and the results sent back to the desktop of the person writing the research paper. This type of use certainly wouldn't be limited to desktops; the information could be displayed on any device with enough bandwidth to receive the data. We refer to this as remote rendering. The end result is that any device with a network connection and a display could be used to view output the device is not capable of producing.

With this new form of visual computing come many new challenges. We need to address issues concerning the interface to the system, resource sharing and allocation, distributed event management, data compression and an adaptable graphics system. Furthermore, current graphics hardware is not well adapted to retrieving a rendered image from the hardware. These are only a few of the many possible problems in doing this correctly. Our goal of this paper is to establish a framework for the future, taking into account the accomplishments that have been made to date, improving on the ones that haven't been successful and incorporating the ones that have. We also describe several usage scenarios that exemplify specific uses for a system of this nature.

2 Past Technology

There have been several research projects over the past few years that address many of the issues we are concerned with. We will mention WireGL and and its successor Chromium since we will be using the latter as the starting point for our research [14, 15, 13]. There have also been numerous projects that focus on parallel rendering other than WireGL and Chromium. We choose not to include them because the underlying rendering infrastructure is not the direct focus of this work. For further information, see [11, 23, 9, 22, 8, 33, 34, 2, 20, 1, 24, 25, 7]. In this section we intend to note a few of the projects that are most applicable to our research. These include work done with display walls [41, 21, 5, 4], as well as some unique projects such as the Office of the Future [28, 3] , Project Oxygen [10], and the Interactive Workspace Project [27, 17, 26].

2.1 WireGL and Chromium

WireGL was developed by Humphreys, et al., as a system for sort-first parallel rendering on clusters of workstations [14]. It focused mainly on adapting existing OpenGL applications for tiled displays. One of the main priorities was image resolution, not processing speed. Because of this, applications running on WireGL could run as fast locally as they could on a cluster of workstations. Since WireGL was originally designed with displays walls in mind, it did not make an effort to take advantage of all of the available resources in a cluster. Another problem was the lack of flexibility in terms of sorting classification. The fact that it was designed for sort-first parallel rendering meant that it could be difficult to load balance an application. Because of this, WireGL relied on the spatial locality of graphics primitives to be able to perform well. Chromium, WireGL's successor, provided more flexibility by enabling sort-last rendering as well as a configurable infrastructure and the possibility to extend OpenGL progammatically.

Chromium is designed for parallel applications and manipulating streams of graphics commands on a cluster [15]. Like WireGL, Chromium operates by replacing the system's OpenGL driver in order to intercept graphics commands issued by an application. Chains of Stream Processing Units, or SPUs, are used to define the operation of a node in a cluster. The SPUs can be arranged in an almost arbitrary fashion, allowing for a more reconfigurable configuration than WireGL could offer. The specific task that each node is responsible for is defined by its own chain of SPUs. Therefore, there doesn't need to be a difference between application, worker, and server nodes. Each use the same underlying libraries, but pass different types of information to each other. For example, an application node may pass geometric primitives to worker nodes, which also act as servers rendering the primitives to their framebuffers and reading back the results to send it on to another server. A Directed Acyclic Graph is defined in a configuration file that defines where nodes receive data from and where they send it. In this way, different parallel rendering configurations can be defined.

Chromium's SPUs can also be extended for additional functionality. In doing so, it is possible to write new extensions to OpenGL and modify the operation of existing commands. Any SPU is, in fact, a re-implementation of a subset of OpenGL commands. For example, to send graphics commands over a network, the Pack SPU re-implements all commands that need to be sent, and instead packs them into a buffer that can be sent as a whole. New commands for parallel graphics, as described by Igehy and Hanrahan [16], are also implemented by creating new OpenGL commands and intercepting them at the driver level. Since SPUs can be chained together, it is also possible to read back an image that has been rendered, modify it in some way, and send the result over the network. This will prove useful for implementing certain compression techniques as described later in section 3.3.

We intend to use Chromium for our continued research as it provides us with the flexibility and power that we will need to be able to implement our proposed system. However, there are several things that Chromium does not yet offer that are important to our system. We see Chromium being useful as a tool for investigation, but recognize that it may not be able to meet all of our requirements. Our goal is to develop a system with enough generality that it can meet all of our needs.

2.2 Display Technologies

Much of the research being done with displays is motivated by the increasing demand for higher resolution. Display walls have typically been the only solution because of the slow increase in average display resolution over the past few years. Specialized high resolution desktop displays are also emerging, such as IBM's T210 and T221, but are still not built for the average workstation. Display walls are the only way to view high resolution data sets. There has been a lot of work done with display wall technology and we address some of the relevant parts here.

2.2.1 Tiled Display Walls

One of the typical ways to achieve higher resolution displays is by tiling multiple displays together. Desktop displays currently have borders that don't allow for continuous display areas so projectors are usually used. Image projections can be placed as close to each other as needed, and even such that they overlap. One of the most difficult problems with display walls is aligning the projected images. Early display wall systems were not capable of automatically calibrating the projected images [36, 12, 39]. Without automatic calibration, projected images need to be manually aligned by adjusting thumb screws for projector mounts, as well as projector focus and zoom. This can be a painstaking and time consuming process and if someone was to bump the setup the entire display needs to be redone. A few papers have been published that have solved the problem of automatically calibrating projected imagery [21, 41, 37]. The typical method is to project fiducials, or structured light, onto a surface and recover the result with multiple cameras. The projected images can be used to define a mapping from the projector to the surface. Alpha masks are then used to seamlessly blend the projections from multiple projectors.

Before systems like Chromium [15] and Princeton's parallel rendering systems [33, 34, 32], display walls were typically run by specialized graphics supercomputers. These could be expensive setups with multiple supercomputers required to run all of the projectors. We see the future of display walls in a much different light, where they have been removed from the rendering power (be it supercomputer or workstation cluster) to be an independent entity. Our vision is that such display devices will act as thin-clients, only responsible for receiving images or graphics commands, and sending interaction events back to the rendering source. This will enable display walls to be located anywhere and in a way that they aren't reliant on a devoted cluster or supercomputer to run them. For more information about graphics hardware and display devices refer to section 3.6.

2.2.2 The CAVE

One particular type of display wall that has received attention in the past few years is the immersive environment. One of the first published systems regarding immersive environments in computer graphics was the CAVE, the Audio Visual Experience Automatic Virtual Environment [5, 4]. The CAVE gained inspiration from displays such as early flight simulators and other virtual reality devices like head-mounted displays. It is a virtual reality interface that was originally designed for scientific visualization. The CAVE attempts to achieve something called "Suspension of Disbelief," where the user is able to ignore the interface and focus on the images being produced. This is the basic idea that all immersive environments attempt to achieve. The goal is to make the user believe that what they are seeing is a true representation of reality.

A CAVE is essentially a room enclosed by rear-projected screens that fill the user's periphery. A user is tracked within the room and the images on the screens are displayed with respect to the user's position, demonstrating the idea of a view-centered perspective. To obtain correct stereo projection, the user's orientation as well as position must be tracked inside the CAVE. This is done using a head-mounted system that allows for calculation of where the viewer's eyes are located. Images are then projected in either active or passive stereo, and are viewed using special eye-wear. In the case of active stereo, an image is projected for the right eye while the left eye is covered by shuttered goggles, and then an image is projected for the left eye while the right is covered. Images need to be generated at twice the framerate as usual because they are divided between the eyes. This can be alleviated by displays that automatically divide an image into two images, one for each eye. When projecting in passive stereo, two projectors are used to project offset images, which are each polarized in a different direction. The user will need to wear polarized glasses that capture the correct image for the correct eye.

Since the idea of the CAVE was presented, numerous variations of immersive environments have been attempted. Most of these build on the shortcomings of the CAVE in the hope of creating a more convincing experience. To achieve this, new projects focus on eliminating the equipment required for tracking position within. Others, such as the next two projects, are trying to move immersive environments into more convenient settings. One of the problems with implementations such as the CAVE is that they require specialized visualization centers just as display walls do. The hope is that these can be eliminated to make the experience more life-like.

2.2.3 The Office of "Real Soon Now" and Office of the Future

With the Office of "Real Soon Now" Bishop and Welch have taken a different approach to computing in the office environment. They have gone away from using traditional CRTs or LCDs and have begun using projectors as their primary displays. They reported the result that they have created a better working and collaborative environment. An added benefit is that of better ergonomics since focusing on the wall at two to three meters is much easier than focusing on a monitor at a half meter. Furthermore, projecting onto a wall facilitates a better working environment in that there is no need to crowd around a monitor in a confined space. Both researchers have found that eliminating a large monitor from their desks frees up enough desk space that having a large desk has become unnecessary. The result is much like having a conference room as an office.

Bishop and Welch also mention several of the problems that arise from their new office configuration. They say that projectors produce a lot of heat and the fans needed to cool them are more than loud enough to be noticeable. Another issue is that the projectors they use are not bright enough to provide high contrast with the room lights on; the lights in the room must remain off so that the projected image is visible. The bulbs used for the projectors also have a short life span and must be replaced occasionally. Because of this and existing commodities of scale, projectors aren't nearly as cost-effective as CRTs, but Bishop and Welch speculate that prices will drop as projectors find more widespread use.

The Office of the Future extends these ideas to remote collaboration. Raskar, et al., are working toward an environment where a corner of an office can be transformed into a spatially immersive virtual environment, building on the ideas presented by the CAVE and other tiled display systems [5, 4]. By replacing the lighting in a room with projectors, they intend to make it possible to use any surface for display. Another aspect of the project includes using structured light techniques for image-based modeling of the office.

The purpose of doing image-based modeling of an office is so that the office can be reconstructed and rendered at a remote site, creating a virtual office space which links several remote offices together. This way, a realistic reconstruction of another office, or multiple offices, can be projected onto the corner of your own office in a spatially immersive manner. The effect is that of looking into a collaborator's office just as if it was part of your own. Raskar, et al., are all trying to avoid using virtual environments such as head-mounted displays which can disassociate a person from his surroundings, making his own office useless. They want to keep the convenient attributes of the office, such as not having to go "down the hall" to use such systems as the CAVE, while adding to the effectiveness of the office. In fact, they want to avoid anything virtual other than the environment itself, making sure not to use such things as virtual objects or 2D avatars. Collaboration, as they see it, should be as natural as personal communication.

The researchers at UNC face several limitations for being able to do much of this work. First and foremost is the problem of computation power. In order to run several displays in an office setting a supercomputer, of several cooperating computers, must be used. This follows closely with the problems we see with the tight coupling of graphics hardware and the display. If the Office of the Future had remote rendering capabilities, they could control the system at a central location, avoiding the clutter, noise, and heat problems associated with having several computers running in a single room. Projectors would exist as autonomous displays being fed imagery from the central source.

The Office of the Future also faces problems associated with latency of wide area networks. They have a situation where they are attempting to send multiple full scene descriptions of the office environment at real time framerates. Their rendering is a two-pass method, rendering the scene first, and then projecting it onto the surfaces being used for display. As a result, they must find ways to simplify the display surfaces and other data being transmitted. If the model is dynamically changing, as most likely it would be if someone is moving around the office, new models need to be created and rendered every frame. Other problems include load balancing, where one projector may be rendering a much more complex part of the scene than any other. Combining this all with the time it would take for image-based reconstruction of the local office and rendering of remote office spaces, they face a hurdle that will be difficult to overcome.

2.3 Ubiquitous Computing in Computer Graphics

In 1991 Mark Weiser published an article in Scientific American that would predict a good portion of the future of ubiquitous computing [40]. The article described three devices that would exhibit the essence of this new form of computing; "tabs", "pads", and "boards". These devices were post-it sized, paper-sized, and chalkboard-sized, respectively, and each would carry out functions similar to their counterparts. Weiser describes ubiquitous computing as the point at which computing recedes into the backgrounds of our lives, becoming an aspect of everyday life that is no longer thought of as novel. Devices such as the ones he described would replace the ones they are modeled after; a post-it note is no longer a post-it note but an active display capable of sending and receiving information. Similarly, chalkboards take on the role of a large, interactive display surface. Since these ideas were first presented, ubiquitous computing has expanded significantly, but still follows closely the predictions made by Weiser.

Today, ubiquitous computing takes on several different forms. One such form is that of wearable computing, where computing devices become a part of our person. The first such device, actually built in 1961, was designed to predict the outcome of a roulette game [38]. Many small devices have been created and theorized since then. They have taken on many forms, from active badges to eye wear, and continue to evolve into more everyday objects. Many are now designed to interact with the user based on situation context. We aren't directly concerned with the topic of wearable computing, but recognize that wearable computers could also act as display servers for remote rendering. The main contribution of wearable computing to our research is the notion of easily accessible computational power from nearly anywhere.

Another form of ubiquitous computing is pervasive computing. The motivating idea is a little different from that of wearable computing. The main goal is to have computational power accessible at anytime and from anywhere. This is more representative of the type of ubiquitous computing applicable to our system. Remote rendering has the goal of providing graphical computational power to any device with a network connection. For this reason, we are looking toward a system that will be able to provide a different type of ubiquity. Few devices will be capable of handling big data constraints in computer graphics in the near future. It would be very useful to be able to access the graphical computational power of a cluster of workstations from any display.

2.3.1 The Interactive Workspace Project

The Interactive Workspace Project at Stanford is one of the few projects in the area of Computer Graphics that deals with Ubiquitous Computing [27, 26, 12, 17, 18]. There are several different components related to the overall system, including the Event Heap, the Data Heap, and ICrafter. Together they make up iROS, the underlying operating system for the iROOM, the Interactive Workspace at Stanford. These components work together to create an environment where multiple devices can communicate with each other, using and controlling different displays and exchanging data. The environment is much like a typical conference room, but outfitted with smart displays and an interactive mural. The interactive mural is a smaller display wall that a user can interact with using a specialized stylus. A conference table in the middle of the room also has a display built into its surface that can also be used for certain visualization needs. The Interactive Workspace is a form of ubiquitous computing in that any user may enter the room and be able to control the displays in the room from his or her laptop. The user can also display the contents of their desktop on any of the displays in the room. Thus, the Interactive Workspace must be able to compensate for devices that are entering and leaving at any moment. The components of the iROS are what make this possible.

The Event Heap is designed to coordinate communication between different devices and applications [17]. Examples of this include controlling a display from a laptop or writing on an interactive screen with a specialized pen. The Event Heap is based on tuple spaces; a specific event is associated with a source and target as well as having other fields attributed to the event. Events can be created through a java applet on a web-page or by individual applications. A central location stores all of these events so that they can be queried to find out if an event has been created for a particular device. Events can be destroyed when they are queried, or can remain for other devices to use them as intended. If events remain too long, they will simply expire and be removed. As such, the Event Heap is the basis for communication in iROS.

The iROS also requires some way to move data from one device to another. Through the Event Heap, a producer can indicate that they want to store data and associated metadata. Similarly, consumers can query for information based on the metadata while also describing the formats they can receive. Using the Data Heap, data is dynamically converted to the format the consumer wants, with the benefit that the producer need not be concerned about how it uploads the information [26]. Several data conversion tools are defined that help convert information from one source to another. It is possible to chain these tools together to find a conversion from one type to another that does not exist by itself. This way, it would be possible to store information from a spreadsheet on the data heap which could be received for use on a word processor. Without such functionality, it would be difficult to integrate different applications for use in the Interactive Workspace.

ICrafter presents a way to create user interfaces for any controllable hardware or software in the Interactive Workspace [27]. Services can publish “beacon” events that describe their service. ICrafter can then query the Event Heap to determine available services. The user is able to ask the iROS Interface Manager for a user interface to control the service. Like all of the components of the Interactive Workspace, ICrafter functions so that services do not need to know what type of device will be controlling them. Thus, when a new device enters the room that may provide a service for other users in the room, all it needs to do is send a “beacon” event to the Event Heap describing the service it provides. ICrafter is then responsible for determining the user interface for that service; one can be defined by the service, or ICrafter can generate a generic interface.

While the Interactive Workspace as a whole presents a good example of how important design decisions can affect the overall function of a generalized system, it still has problems of its own. The Interactive Workspace wasn’t designed as a solution for complete ubiquitous computing, but does overcome some of the issues related to this problem of the tight coupling of graphics hardware and its related display. With this system, any user is able to display information anywhere they choose. However, the Interactive Workspace does not fully use the power that graphics hardware has to offer. A solution for graphically intensive applications must be of a different nature. The problem of big data constraints is not fully addressed by the Interactive Workspace. A user does not have the option of viewing extremely computationally intensive applications on a laptop or other device. The Interactive Workspace would be one research project that could benefit from remote rendering as well.

2.3.2 Project Oxygen

The motivation behind Project Oxygen is to target computation toward people rather than machines [10]. The attempt is to move away from expensive, high-power computers and toward computers that serve people's lives. Project Oxygen is a form of pervasive computing in that allows computation to be freely available everywhere. It is also more of a vision than a system. Currently, many of the necessary components exist but are not unified as a whole. The project is also an experiment in more natural interaction with computers through voice communication and gestures. As such, this project is much more inclusive than others in the area ubiquitous computing.

There are many different aspects of Project Oxygen that need to work seamlessly to create an environment capable of what they are trying to achieve. These include device, network, software, perceptual, and user technologies. New devices need to be created to facilitate this sort of ubiquity. They will provide the source of computation and communication for users and need to be available everywhere. The central idea of Project Oxygen is that these devices should not be cumbersome to use. They can also take on different roles, as embedded, handheld, or other types of devices. All devices need to be able to communicate through some sort of network. If, for example, you were to walk up to an Auotmated Teller Machine and it automatically knew your name, the device would need access to a database of all people in the system. Many problems may arise if hundreds or thousands of these small devices are trying to access information over a network at the same time. There will also be issues with security that need to be dealt with in this environment. Project Oxygen currently has technology to alleviate some of these problems, such as the Self-Certifying and Cooperative File Systems for secure data access.

Those involved with Project Oxygen have also started work a GPS system for indoor use. Many of the devices they see as being usable in a ubiquitous computing environment would need to be tracked so they can be aware of their context. Cricket, their location system, uses ultrasound and RF signals to communicate with devices. Currently, the transmitters are too large to be used easily with handheld devices, but as the technology improves, this will become much less of a problem. Software also needs to be developed to allow these devices to communicate with one another. MetaGlue provides the communication infrastructure between these devices and the controlling systems. It allows users to interact with software and data from anywhere by detecting new devices and allowing devices to pick up previously established connections.

Many of the components of Project Oxygen have been completed or are being worked on, but the vision is far from complete. There is currently an Intelligent Room implementation that is much like the Interactive Workspace described above. Unlike the Interactive Workspace however, Project Oxygen is geared toward an environment that is entirely ubiquitous. Project Oxygen also has a few elements that are more similar to the system that we envision than the Interactive Workspace. Specifically, we are more interested in the ability to have graphical computational power available anywhere. Project Oxygen isn't targeted solely toward issues posed by computer graphics, so it does not take into account problems posed by big data constraints that are central to our system.

2.4 Remote Computing

Two popular systems that have addressed the issue of remote computing are the X Window System, originally developed at MIT [35, 30, 29], and Virtual Network Computing from Olivetti & Oracle Research Laboratories (ORL) which has since been acquired by AT&T [31]. The motivating idea behind both is quite similar to ours; applications are designed to be displayed locally but in many instances it would be convenient to be able to run the application from a different location.

Some of the first forms of remote computing were achieved through "teleporting" as described by Richardson, et al., using the X Window System as the underlying infrastructure [30, 29]. When first created, teleporting was concerned with problems arising from dealing with different server display configurations, such as framebuffer depth and frame size. We have since moved to a more heterogeneous computing environment, where such issues are of little concern. Current solutions are more technologically advanced and are dealing better with problems of bandwidth and client state. The VNC viewer is a good example of this. It functions as an ultra-thin client, only responsible for receiving graphics commands, displaying them, and sending back interaction events.

The motivations behind VNC are very similar to the system that we envision. Of particular interest is the idea of the thin-client, a client which is not responsible for maintaining its own state, and does not need to be concerned with the underlying function of the system. As a result, if a VNC session is closed, it can be started up again in the same state, from the same location, or from another computer. The VNC project grew out of the Videotile experiment at ORL, a display device with an LCD screen, a pen, and an ATM connection. The Videotile was originally designed for displaying movies. While it functioned as intended it required a large amount of bandwidth. In the end, bandwidth issues prompted the current state of VNC. To reduce bandwidth, Richardson, et al., made the observation that most of a typical desktop is taken up by blocks of the same color, motivating the ability to be able to define large portions of the screen at one time. The simplest graphics primitive in VNC is defined as a rectangle of pixels at a specific location. The VNC researchers also note other important aspects of the system that allow them to reduce the amount of information that needs to be sent to the client. Examples of this are only updating pixels which have changed color since the last update, and providing the ability to move entire rectangles of the framebuffer to a new location. With these improvements to the Videotile, VNC has migrated to its current state.

Certain aspects of both X and VNC make them inherently different from the system we envision. One of the differences with VNC is that it doesn't address having multiple clients connected to the same machine. X provides for this, but doesn't encompass the idea of parallel computing. Our system needs to be capable of both allowing multiple users, and enabling them to run applications across multiple machines. Also, X and VNC are both geared toward running a remote desktop but don't perform as well when running graphically intensive applications. Issues of data compression and enabling efficient image data transfer are central to the work we are proposing. We also need to be concerned with overcoming big data constraints that would render X and VNC unusable. As a whole, X and VNC don't provide quite the right tools to enable remote rendering.

3 System Layout

In general, little work has been done in the areas of remote rendering and visualization. Software such as VNC and the X Window System enable remote visualization, but none of these are geared toward computationally intensive computing, 3D graphics, and big data constraints. There are many aspects of remote rendering that need to be addressed for this to be possible. Technological improvements are needed in graphics hardware, display technologies, and networking. We also need to expand research in the areas of resource sharing and allocation, data compression, dynamic and adaptable networking systems, human computer interaction, and distributed event management.

3.1 Central Computation Power

We intend to use Chromium as the rendering power for our system. Chromium will be a useful tool that will enable research in the area of remote visualization. Chromium is currently capable of running parallel applications and performing parallel rendering, but there are many improvements that are required to make remote rendering attainable. Specific improvements are explained in depth below.

First and foremost, the system used for the underlying rendering power needs to be capable of overcoming big data constraints. The pattern we have seen with many current technologies is that they do not provide a method for accessing power outside that capable of a particular device. Ideally, any user should be able to access the rendering power of a cluster from any display. We will also require the ability of parallel rendering, such that an application can be rendered to a tiled display. Without these requirements, remote rendering degenerates into a simple remote computing problem, which as described has already been solved by systems such as the X Window System and VNC. Furthermore, the power of the rendering system must be accessible through a standard interface such that applications need not be modified to run on the system, and new applications can be written easily. The system must be scalable and flexible enough to perform many different types of tasks. Chromium already provides these abilities, so it is a logical choice for a starting point.

The parts of the system that Chromium does not currently provide are described below. These include resource allocation, data compression, adaptable graphics, and the system and application interface. We have already completed an event distribution system, and that element is described below as it is an integral part of our system. At the end of this section, we also describe new display paradigms that we believe will arise from a system such as ours.

3.2 Resource Allocation

One of the requirements of our system is that it must be able to simultaneously handle multiple users. The rendering source must be capable of allocating resources to different users in a specified manner. This may depend on the application the user is running, the current load on the entire system, and the bandwidth available to the user. Thus, the rendering system must be fully aware of what it is doing, how busy it is, and network latency to different clients. The rendering source will almost certainly have a limited amount of resources available to it. If there are multiple users on the system, we must be able to determine priorities for each user and the amount of resources that each user requires.

Our system must have an application interface to a resource manager that has knowledge of all of the applications running on the cluster. Some applications, mainly parallel ones, will not be able to run with fewer than some number of worker nodes, while some may be scalable and can require a variable number of nodes. The interface should then allow for applications to specify a minimum number of nodes with which it can operate and whether or not the application itself can benefit from having more. Chromium is able to support more rendering nodes to any application through tiling the output, so it is often possible to improve the rendering performance of an application. The caveat is because there becomes a point at which the size of tiles and the amount of overlap of geometric primitives between tiles becomes so high that adding more rendering nodes will not improve performance. It is important to note the difference between parallel applications and parallel rendering. Chromium allows for parallel rendering of any OpenGL application, but not all OpenGL applications are parallel. Chromium provides parallel extensions to OpenGL as described by Igehy, et al. [16]. An application must be written to explicitly take advantage of parallel computation in the general sense.

The resource manager must also be network aware for similar reasons. If adding more nodes requires too much network bandwidth, it should be able to determine whether or not the whole system will benefit from this. This awareness should apply to the local area network and also to outgoing bandwidth to users. This ties in with the Section 3.3 which describes the ability to perform data compression for these cases. The resource manager should also be aware of the case where the bandwidth available to the end-user is small enough that compression alone cannot solve the problem. In this case, more worker nodes will not provide better framerates, and it may be possible to reduce the number of nodes allocated.

A target framerate should also be specified that would be applicable to a heuristic for resource allocation. The resource manager would then be able to monitor the framerates for individual applications so that it could determine which applications require more nodes and those which could operate with fewer. It would also be desirable to refrain from changing the number of nodes allocated to an application when possible as this may have adverse effects on the viewing experience. A constant framerate would be much better than a widely varying one. Also, the resource manager would need to be aware of the case where it simply cannot provide nodes upon request for a new application and will notify the user in such a case.

These are the main aspects that we must consider when implementing a resource manager. Without such a tool, it will be very difficult to manage a system that allows for multiple simultaneous users. If remote rendering on a wide scale is to be a possibility, it cannot survive without this element. Many aspects will need to be analyzed with regard to when certain actions are necessary, such as tradeoffs between adding new nodes and the added network traffic that results.

3.3 Data Compression and Adaptable Graphics

Concerning the required bandwidth for graphical applications, we will need to address the issue of data compression to make our system a reality. In Section 3.6 we describe two categories of displays that will be capable of receiving data in different forms. In the first case, a display only needs information about the image it needs to display in the form of pixel values. Sending a whole framebuffer to a device would be a very bandwidth intensive task though. For this reason, we need to look at different ways of doing image compression. There are already many compression algorithms that we will be looking at, including streaming compressions such as Mpeg4, and single image compressions like JPEG and PNG. Furthermore, we would like a compression algorithm that is capable of adapting to the traffic in a network, so that it is performing more aggressive compression when the network is saturated. Of course, the type of compression will also depend on the speed at which it can be performed, with the requirement that it slow down the framerate as minimally as possible, if at all.

The second type of display will receive buffered graphics commands. Currently, Chromium supports both types of displays, but does not attempt to do compression in either case. We want to be able to perform compression for whatever type of display will eventually be receiving data. With this second type of display, we want to investigate performing different types of compression, such as geometry compression as described by Michael Deering [6]. This is done by reducing the precision used for floating point representations of positions and colors, using a lookup table for normals, and converting triangle data to a generalized mesh. Deering was attempting to address the problem of input bandwidth to graphics accelerators. We would like to extend this to reducing the amount of bandwidth required for sending data over a network.

There are several aspects of compressing data that we will need to analyze to determine what the best compression algorithm will be for our purposes. For any image compression, we will need to look at the effects on compressing tiled images and their appearance after decompression. Visual artifacts will probably be produced at the seams in a tiled image that will be undesirable. For these reasons, it would be extremely beneficial to find a lossless compression. However, a lossless compression would not have the benefit of being able to adapt to the amount of network traffic. Finding a method that provides us with the ability to present lossless or lossy images would certainly be beneficial. We will also need to analyze the effect a compression algorithm has when added to the end of a rendering operation. If it is the case that a rendering node can render an image, compress it, and transmit it before it receives the next frame, then we won't have to worry about the effect of compression on framerate. However, for larger image sizes it may be the case that compression will decrease the rate to a point that it would have been easier to send the unmodified image in the beginning. To adjust for these possibilities, we need a graphics system that has the ability to adapt to a given situation.

When compression is slowing down framerate in an undesirable manner, an adaptable system must be able to compensate. There are several different options available, and how they are specified is just another element of the system that we must determine. It may be that image size would have to be dynamically reduced to compensate for issues related to compression time or bandwidth problems. In the case of high network traffic, it may be the case that a more aggressive compression can be applied to create less data flow. However, if compression times are unbearably high, the system must be able to realize this and allocate more resources for the purpose of data compression. New nodes could be allocated for the sole purpose of compressing completed frames. It also may be that the number of allocated resources is inadequate for the application and more will be required. These are only a few of the possible situations that may arise in a system of this type. There may be others not discovered until we are well into implementation.

3.4 System and Application Interface

The overall system will need an interface such that a user can begin an application on the cluster. The simplest interface could be a web page, with the benefit that software would not need to be installed on the user's system. Most web browsers have the capability of displaying streaming media or deferring such a task to a helper application. When a user calls up an application on the cluster, the resulting output could be compressed into a standard format and streamed to the browser or helper application. However, in this case, it would be difficult to interact with the application by sending events back to the cluster. Proprietary applications have no concept of the Chromium libraries for event distribution, so interaction can not be done through them. As described in Section 3.5, the event distribution system is removed from the rendering such that interaction is still possible. Another web page could be provided with an interface for sending events, but this has limitations of its own. The user would now be interacting with one window and viewing the results on another. Applications could not be viewed in full-screen mode, and users on devices such as PDAs have too little screen real estate to have two windows open. For this reason, custom applications would also have to be created to allow for user-interaction.

The interface application would act as both an interface to the system as well as a viewer for output from the application running on the cluster. This way, all Chromium libraries can be accessed by the interface, and we also allow for customization based on the architecture of the device being used for viewing. Interfaces can then be written for all platforms, including personal computers, PDAs, and any other device used for display. On startup, the user could be presented with a screen through which they would define some preliminary parameters, such as the number of nodes requested. The system interface would then call up an application, and defer to the interface for that application, which is then responsible for displaying images and distributing events.

For the displays that we describe in section 3.6, one of the added benefits is that the user interface has nothing to with the display device itself. This would essentially eliminate the requirement for an application interface as previously described. The interface can be defined in the application so the display has no concept of what the user is interacting with. If the display device is aware of the location of the cluster, it can call up an application which displays an interface as well. The display only needs to notify the application of the location of the event.

3.5 Event Distribution

One of the key components in any truly interactive system is the ability to generate events based on user input. The Event Heap component of the Interactive Workspace project at Stanford is one of the few projects that have addressed the issue of distributed events [17]. We have already begun work on a distributed event system for Chromium, called CRUT (the Cluster Rendering Utility Toolkit for Chromium), that is loosely based on some of the ideas of the Event Heap. An Alpha release has been included with the last two versions of Chromium. CRUT makes it possible to program a user-interactive application to run on Chromium. It is modeled after GLUT to make it easier to learn and to better enable migration of GLUT programs to the Chromium environment [19].

CRUT is an API for Chromium that allows for sending events from an event server back to the client application. We have made an effort to make the CRUT libraries as flexible as possible so that users are not bound to a certain graphical toolkit. A custom GUI can be implemented that only needs to call functions in the CRUT server library to send events to the application. Thus, other toolkits such as Motif or Tk can be used to create a GUI that Chromium can render into that just needs to use the CRUT library for sending events. We also give freedom to the application by allowing it to retrieve events in different manners. We don't want to bind an application to using a main loop similar to `glutMainLoop`. Applications written for Chromium are of a different nature than many desktop applications, and may require the freedom to maintain close control of their execution. An application may poll for events in the same way as provided by the X Windows System. A main loop is also provided so that programs can run as they would under `glutMainLoop`, but by allowing for event polling, we restrict CRUT from binding programmers to a specific programming model.

Another important benefit of CRUT is that it is completely removed from any rendering context in Chromium. This allows a dedicated event server to be defined that is not associated with the rendering in any way, but has the purpose of generating events for the application. This could be extremely beneficial for controlling an application meant to render to a display wall. With the current notion of a display wall, there is no single rendering node, so it is unclear how one could interact with the application. By providing a separate server, one could use a dedicated computer placed in the same room as the display wall that would allow a user to interact with the application while viewing its output.

One of the main problems that CRUT must address is the inherent latency in distributed computing that prevents events from being processed immediately. For individual events, such as a key press or a mouse button click, this will not be apparent, but when generating streams of events, the latency will be quite perceptible. Currently, the client library for CRUT buffers all of the events it receives. However, events can be generated at a much faster pace than they can be received and processed. For this reason, it will be necessary to look at certain types of events and reduce the number that are sent across the network. We are currently looking at different ways of doing this, including simply sending events at certain intervals, sending only the first and last events for actions, such as mouse motion, and adding an expiration time to events. We hope that in the final implementation, CRUT will also be able to determine the latency in the system and adapt the number of events that are being sent to reflect this.

3.6 Graphics Hardware, Display Devices, and Data Transfer

We address the topics of graphics hardware, display devices, and data transfer together here because of their reliance on one another in our system. As stated, we look forward to a future where the graphics hardware is not designed solely for the purpose of displaying an image locally. We currently have the ability to read back rendered information, but typically at a rate much slower than we were able to create it. We see two necessary improvements to current graphics hardware that will greatly aid remote visualization: increased bandwidth to graphics hardware, and improved readback abilities. Graphics hardware, however, is a small variable in the remote visualization equation.

In Chromium, displays take on the role of servers (or consumers), receiving information from the rendering source, and sending back events. There are two possibilities for sending information to the display device, depending on whether or not you are sending actual image data or graphics commands. In the first case, the display device is an extremely thin client that only receives frames of images for which it is responsible for decompressing and displaying. The only data transferred to the client are pixel values. The display would not be responsible for performing any rendering, but must still have some amount of computation power so that it can decompress the image and control the display. These thin clients could take on many forms. One example would be as tablets that would be clipboard-sized and could be used in a similar manner, for providing patient information in a hospital for instance. Also, as display technologies improve, displays are becoming smaller, and include some as thin as a few sheets of paper. These displays will probably not have a lot of hardware associated with them, but, given a network interface, they could receive imagery generated from another source.

The second type of display device is one that is similar to the first, but is also outfitted with graphics hardware capable of rendering incoming streams of graphics commands. The advantages would be that it would put little strain on the central computing source, it may require less in terms of network bandwidth, and a parallel application could run on the server, sending graphics commands to a remote tiled display. One result would be that computers do not require expensive graphics hardware. As graphics commands are intercepted, they are sent over the network rather than to the display driver. The computer then acts more like a web server than anything else.

One of the main motivations for remote rendering is being able to visualize datasets in a convenient environment, such as the desktop computer or handheld device. Apart from these conventional displays, we think there are other ways in which remote rendering could be useful. One new type of display possible would be a movable, reconfigurable display wall. Once a display has been decoupled from the graphics hardware, it becomes unnecessary to have dedicated visualization centers such as those currently used. Display walls could be placed in conference rooms, classrooms, or even in the office. Another type of display we think will emerge is a tablet-like display which might be used in settings such as a hospital, or any other environment where it would beneficial to have this mobility. If fact, any display device should be usable for remote rendering, these are only a few of the possibilities.

4 Usage Scenarios

With any new technology, the immediate question that must be asked is whether or not it will be beneficial enough to justify its undertaking. We believe that there are many uses for a system of this nature. In this section, we outline some examples of usage scenarios in the attempt to show that our system will not simply be a solution for a particular niche, but that its application is indeed broad in scope.

4.1 The Power of Graphics in the Field: Virtualizing a Battlefield

One place that we see our system as being beneficial is in any field setting where computational power is severely limited. This applies not only to a battlefield, but also to other areas, such as anthropological, geological, or environmental research locations. A scientist on an anthropological dig may benefit from being able to view data representative of where fossils are buried, but lacks the means for doing so. If data of this nature was gathered using sonar or radar scans, the information could be compiled at a central computing location and sent to the person in the field. It could then be used to target specific locations to unearth. Similarly, an environmental researcher may be able to use information about aquatic contaminants on an interactive map. In both of these cases, if the displays being used were integrated with a GPS unit, it would make it a simple task to correlate the data and the current location of the display, allowing the user to better find what they are looking for.

We see the situation in a battlefield to be more constrained than the others, and as such provides a better example of the advantage of having a system such as ours in place. The matter of safety helps describe the benefits of having portable access to intensive graphical computation power. For military personnel in combat situations, imagery could be streamed to a device carried by a designated member of the ground troops. For example, in a situation where troops are infiltrating an enemy building, important information regarding the building layout and whereabouts of the inhabitants could be displayed on the device. Information could be collected via infrared and radar scans from aircraft flying at a safe distance above the building. The information is collected and compiled on a cluster back at the base, and a three dimensional representation of the building, including information from the infrared scans to determine the locations of people inside the building, is sent to the ground troops. The troops would be able to interact with the images streamed to the display, allowing immediate knowledge of the layout of the building.

The obvious advantage is being able to infiltrate the building with knowledge of the exact location of inhabitants, leading to increased safety for troops engaged in the operation. A concrete plan for infiltration can be formulated and acted upon. Essentially, the guesswork has been taken out of a potentially hazardous operation. There are many other military applications of such a system as well. Displays such as the one described do not need to be handheld; they can be placed in aircraft and ground vehicles, and used for such purposes as mission briefing. Portable display walls could be taken down and erected as needed, with access to the same rendering power previously described.

4.2 Interactive Education

One of the current trends in research work for graphical displays is aimed at creating lighter, cheaper, bendable displays. LCDs are already beginning to replace CRTs as the primary type of display being sold. Paper-like displays are taking the progression one step further. We now have displays that are as small as a few millimeters thick, and the technology is only getting better. One place where these types of technology could be applied is in the classroom or office.

Our usage scenario here describes a classroom that has been fully integrated with remote rendering capabilities. A cluster in the basement can perform graphical computation for the entire school. Each classroom can be outfitted with multiple displays, each performing different functions but all geared toward the general purpose of facilitating education. The first type of device that could be used is a roll-away display screen in the front of the room. It would be used much like the projection screens popular in many classrooms today. The main difference is that the display is active; it is essentially a roll-away display wall. The teacher or lecturer could control the output on the display from either a control console, which could be another display built into a podium, or through an interaction device similar to the one used for the Interactive Workspace [18]. The teacher could also delegate control of the large display to a particular person from the control console. In this way, a more interactive environment can be achieved as students become more involved in the learning experience. This would also require another type of display device available to each student..

Student desks could be integrated with a display that would also be managed from the control console. The teacher would be able to define the information displayed on each of the students' individual displays. For example, if the setting were a chemistry classroom, a large molecular model could be displayed on the large display screen and the teacher could zoom in to a particular part of the model for display on each of the students' displays. As mentioned above, the teacher could delegate control of the large display to one of the students, possibly requesting them to define a particular part of the model. The student could then be presented with an interface to the large screen on his own display, but no one else would be able to control it. Interacting with his own particular display, the output on the large screen would change as he wants for the whole class to see.

Having multiple displays receiving the same output in this manner would require the use of a proxy server for the classroom. This would function as one render server that the cluster sends information to, which is then responsible for distributing the images to the multiple displays. Without such a server, the cluster would be responsible for sending multiple copies of the same images, requiring a large amount of bandwidth leaving the cluster. The proxy server acts as a single recipient of the data which is on both the same network as the cluster but also a local subnet for the classroom.

4.3 The High-Tech Hospital Ward

One environment always in need of easily accessible information is a hospital. Doctors and nurses need to be able to keep track of multiple patients' data in an easily manageable manner. The common method of doing so is having clipboards with relevant information at each patient's bedside. One of the problems with this is that certain types of information, such as X-rays, MRI scans, or ultrasound data, can not always be viewed at the bedside. A hospital worker could benefit greatly from the ability to access information for multiple users from a handheld tablet PC or PDA.

A central database of user information could be kept on a cluster, allowing access to devices throughout the hospital. The need to manage clipboards of patient data and the requirement for paper-based information would be eliminated. Graphical information related to each client would also be stored on the cluster, accessible in the same manner as all other related data. As new patients arrive, new database records could be created for them, and any new information could be added to the record as produced. Historical information could also be kept on record, allowing doctors and nurses to find information regarding allergies or other health complications when the patient is not able to provide the information himself. Multiple people would have access to the same data at the same time and from different locations. Access would not necessarily be restricted to people located within the hospital.

Remote rendering would also make it possible to contact specialists in particular fields to acquire better analyses of patient's conditions. The information can simply be streamed to the specialist, who can then use it to determine a diagnosis for the patient. Since the interface to the overall system can be as simple as a web page, the specialist would not have to download any new software. For some particular applications, it may be easier to interact with an application if software is downloaded, but it should not be entirely necessary. This allows for easy collaboration much like that described in section 4.5.

This type of system could also benefit medical research, a field where it is normal to produce high-resolution volumetric datasets. A visualization center may not always be the easiest way to view such data, and in some cases this information may need to be viewed as quickly as possible. Remote rendering would provide a solution for viewing these large datasets at an individual's desktop or on some other display device. Furthermore, information regarding the user of the system can remain stateless much like with VNC [31]. This would make it possible to leave one's workstation with an application still running on the cluster and move to a colleague's office to bring up the same application. Thus, remote rendering would not only be beneficial to workers in a hospital environment, but to a greater part of the medical profession as well.

4.4 Scientific Visualization

One of the main motivations for remote rendering is the area of scientific visualization. The problem with the way scientific visualization is currently achieved is accessibility. We have mentioned several times the concept of the specialized visualization center. Often, display walls require their own dedicated space as they are too big to be housed in most convenient locations. As a result, both the display wall and the rendering source must be located outside the office setting. This inconvenience means that the visualization center can become largely unused.

Most applications in scientific visualization require more rendering power than an individual PC can offer. Remote rendering strives to make scientific visualization more accessible, bringing the aggregate power of a cluster of workstations to the desktop setting. A reasearcher in his office should be able to use the resources of a visualization center remotely. Of course, on a PC, it will not be possible to view datasets at the high resolution offered by a display wall, but smaller tasks will be made easier. Furthermore, with the advent of new display paradigms as described in section 3.6, it will be possible to house a smaller display wall in the office, providing all the capabilities of a dedicated visualization center in an easily accessible manner.

Remote rendering also allows for the sharing of resources in a visualization center. Currently, when one researcher is using a display wall, others are not able to access the cluster. Remote rendering will provide for multiple concurrent users. This way, one researcher may be using the display wall while another is accessing the cluster from their office, and others are using the cluster from hundreds of miles away. Thus, remote rendering also facilitates collaboration. It allows access to complex data sets and applications which can be difficult to share or distribute.

4.5 Interactive Collaboration

Many current research projects like the Interactive Workspace and the Office of the Future would also benefit from the capability of remote rendering. In the case of the Interactive Workspace, graphical data from outside the iROOM could be brought in for display on any of the screens, and remote rendering could be used as the driving force behind the rendering work being done for the iROOM. The Interactive Workspace is designed to incorporate many different display devices, few of which would have the abilities to render large complex models or intensive graphics applications. Small devices, like tablet PCs and PDAs, would be able to take advantage of the rendering power a cluster has to offer. With the current design of the Interactive Workspace, it shouldn't be difficult to incorporate remote rendering either. A designated server for the cluster could broadcast the beacon events to the Event Heap defining the remote rendering service. ICrafter could be used to create an interface to it, and the DataHeap could potentially be used for image retrieval.

The Office of the Future would benefit from remote rendering as well, but in different ways. In effect, the Office of the Future is trying to achieve many of the same tasks as remote rendering. The Office of the Future faces many problems having to do with bandwidth, graphical complexity, and brute rendering force. Data compression techniques as those described in section 3.3 could be used to alleviate the problem of sending full scene descriptions across a wide area network. Since the Office of the Future currently uses graphics supercomputers for rendering power, the project would be able to take advantage of the scalability provided by a adding extra worker nodes to the cluster. When using a supercomputer, it can be both expensive and difficult to expand the capabilities of the current system. Furthermore, having multiple projectors in an office run by supercomputers will create not only a lot of clutter, but also a large amount of heat.

Our system should also be able to perform interactive collaboration tasks by itself, and there is no reason to use it only as an improvement to other research projects. One of the nice qualities of remote rendering is that state does not need to be tracked by the display device. Much like with VNC, applications could be left running on the cluster while the user moves locations to bring up the application in a different place.

Remote rendering extends this idea further. Where applicable, a data stream can be sent to certain displays from a controlling display as in the classroom scenario above. With VNC this isn't suitable because the purpose of VNC is somewhat different, being aimed at remote desktop use. There is really no point in having a desktop streamed to multiple computers.

4.6 Graphical Computation as a Marketable Resource

With the advent of a new technology, new avenues for business opportunities inevitably follow. This usage scenario describes the way in which we could treat visual computing as a resource, much like any natural resource. As such, it can be marketed, and a fee can be charged for its use.

Previous usage scenarios have described the possibility of using portable display walls as render servers. Similar ideas are also applied here, but in a little different manner. Displays could be used for advertising in commercial malls or individual stores. The result would be a new model for advertising. Touch-sensitive devices could be used so that anyone could walk up to the display to interact with it. Entire catalogs could be available at the user's request, who could even be able to order from the catalog, creating an ATM-like use of the display. These displays should not be limited to commercial purposes either. They could be used as a form of information distribution, providing important data to a particular audience.

Most people have had the experience of being handed fliers advertising for a particular product or service. As the size and cost of displays decreases, these handouts may be able to take the form of small displays, much like the "tabs" or "pads" previously described by Weiser [40]. Economies of scale will not be in place for these displays for some time, but they will be a possibility in the near future. Thin, paper-like displays will have no graphics hardware associated with them, so they will need to be able to receive information from an external source. Remote rendering provides a perfect fit, also allowing for arbitrarily complex data to be displayed on devices with no concept of graphical computation whatsoever. All that is needed is a network connection.

Another such display could be used to replace current billboards now ubiquitous along highways and in highly commercial areas. Currently, advertising space such as that for billboards is sold or rented. Similarly, remote rendering could be marketed as a service which would be paid for, resembling a marketing strategy similar to cell phone companies charging for air time. In the case of remote rendering, the marketable resource could be computation time, the number of worker nodes required, bandwidth consumed, or some similar aspect of cluster rendering.

5 Conclusion

Currently, many fields that have requirements for ultra-high resolution datasets lack a convenient way to access them. Display walls provide one way to view large data sets at high resolution, but aren't easily accessible, nor are they available to everyone. Specialized visualization centers are usually built to house these displays, often in inconvenient locations far from the normal office environment. The goal of remote rendering is to make the power available in a cluster of commodity workstations available to more people in a more convenient manner.

We have shown that there are many applications for a system providing remote rendering. Not only are there many new applications, but remote rendering can be used to improve many existing research projects as well. The purpose of this paper has been to provide a description of aspects involved in creating a system of this nature, as well to exemplify the ways in which it can be used. We believe that remote rendering is far from a niche solution to a particular problem, but provides a general framework that can be applied to many different situations.

References

[1] K. Akeley. Realityengine graphics. In *Proc. of SIGGRAPH-93: Computer Graphics*, pages 109–116, Anaheim, CA, 1993.

[2] Kurt Akeley and Tom Jermoluk. High-performance polygon rendering. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 239–246. ACM Press, 1988.

[3] Gary Bishop and Greg Welch. Working in the office of "real soon now". In *IEEE Computer Graphics and Applications*, pages 76–78, 2000.

[4] 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 the 20th annual conference on Computer graphics and interactive techniques*, pages 135–142. ACM Press, 1993.

[5] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The cave: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, 1992.

[6] Michael Deering. Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 13–20. ACM Press, 1995.

[7] Matthew Eldridge, Homan Igehy, and Pat Hanrahan. Pomegranate: a fully scalable graphics architecture. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 443–454. ACM Press/Addison-Wesley Publishing Co., 2000.

[8] David A. Ellsworth. A new algorithm for interactive graphics on multicomputers. *IEEE Computer Graphics and Applications*, 14(4):33–40, 1994.

[9] John Eyles, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, Nick England, and Lee Westover. Pixelflow: the realization. In *Proceedings of the 1997 SIGGRAPH/Eurographics workshop on Graphics hardware*, pages 57–68. ACM Press, 1997.

[10] MIT Laboratory for Computer Science. *Project Oxygen*.

[11] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, and Laura Israel. Pixel-planes 5: a heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 79–88. ACM Press, 1989.

[12] Franois Guimbretire, Maureen Stone, and Terry Winograd. Fluid interaction with high-resolution wall-size displays. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 21–30. ACM Press, 2001.

[13] Greg Humphreys, Ian Buck, Matthew Eldridge, and Pat Hanrahan. Distributed rendering for scalable displays. In *Proceedings of Supercomputing 2000*, pages 60–60, 2000.

[14] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702. ACM Press, 2002.

[15] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702. ACM Press, 2002.

[16] Homan Igehy, Gordon Stoll, and Pat Hanrahan. The design of a parallel graphics interface. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 141–150. ACM Press, 1998.

[17] Brad Johanson, Armando Fox, Pat Hanrahan, and Terry Winograd. The event heap: An enabling infrastructure for interactive workspaces. Technical report, 2001.

[18] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright: experience with flexible input redirection in interactive workspaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234. ACM Press, 2002.

[19] Mark J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface*, August 1995.

[20] David Kirk and Douglas Voorhies. The rendering architecture of the dn10000vs. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 299–307. ACM Press, 1990.

[21] Kai Li, Han Chen, Douglas W. Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Timothy Housel, Allison Klien, 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. In *IEEE Computer Graphics and Applications*, pages 671–680, 2000.

[22] Steve Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. Technical Report TR94-023, 8, 1994.

[23] Steven Molnar, John Eyles, and John Poulton. Pixelflow: high-speed rendering using image composition. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 231–240. ACM Press, 1992.

[24] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinitereality: a real-time graphics system. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 293–302. ACM Press/Addison-Wesley Publishing Co., 1997.

[25] Satoshi Nishimura and Tosiyasu L. Kunii. Vc-1: a scalable graphics computer with virtual local frame buffers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 365–372. ACM Press, 1996.

[26] Shankar Ponnekanti, Brad Johanson, Emre Kcman, and Armando Fox. Designing for maintainability, failure resilience, and evolvability in ubiquitous computing software.

[27] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A service framework for ubiquitous computing environments. *Lecture Notes in Computer Science*, 2201:56–??, 2001.

[28] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry 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 Press, 1998.

[29] Tristan Richardson. Teleporting: Mobile X sessions. *The X Resource*, 13(1):133–140, 1995.

[30] Tristan Richardson, Frazer Bennett, Glenford Mapp, and Andy Hopper. Teleporting in an X window system environment. Technical report, 1993.

[31] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[32] R. Samanta, T. Funkhouser, K. Li, and J. Singh. Sort-first parallel rendering with a cluster of pcs. In *SIGGRAPH 2000 Technical sketches, August 2000*.

[33] Rudrajit Samanta, Thomas Funkhouser, and Kai Li. Parallel rendering with k-way replication. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 75–84. IEEE Press, 2001.

[34] Rudrajit Samanta, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *Proceedings 2000 SIGGRAPH/EUROGRAPHICS workshop on on Graphics hardware*, pages 97–108. ACM Press, 2000.

[35] Robert W. Scheiffel and Jim Gettys. X window system. 1992.

[36] Daniel Schikore, Richard Fischer, Randall Frank, Ross Gaunt, John Hobson, and Brad Whitlock. High-resolution multiprojector display walls. In *IEEE Computer Graphics and Applications*, pages 38–44, 2000.

[37] M. Steele, S. Webb, and C. Jaynes. Homography monitoring and correction of geometric distortion in projected displays. *Central European Conference on Computer Graphics, Visualization, and Computer Vision*, 2002.

[38] Edward O. Thorp. The invention of the first wearable computer. In *ISWC*, pages 4–8, 1998.

[39] University of Minnesota Computer Science Department. *The PowerWall*.

[40] Mark Weiser. The computer for the 21st century. *Scientific American*, 265 (3):94–104, 1991.

[41] Ruigang Yang, David Gotz, Justin Hensley, Herman Towles, and Michael S. Brown. Pixelflex: a reconfigurable multi-projector display system. In *Proceedings of the conference on Visualization 2001*, pages 167–174. IEEE Press, 2001.