Computational Displays

On Enhancing Displays using Computation

Thesis submitted in partial fulfillment
of the requirements for the degree of

Doctorate of Philosophy
in
Computer Science and Engineering

by

Pawan Harish
200507007
harishpk@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500032, INDIA
March 2013

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Computational Displays: On Enhancing Displays using Computation" by Pawan Harish, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. P. J. Narayanan

To God, Rishis and My Parents

Acknowledgments

I would first and foremost like to thank my adviser, Prof. P. J. Narayanan, for taking time out of his busy schedule to guide me along this work. He guided me throughout the entire range of problems encountered in this work - from the conceptual, the algorithmic to the practical and the system design. He helped me in designing the various systems along with the theory behind them while keeping in sight the overall goal and flavor of the research domain. I am grateful for his contributions to this work and even more so in guiding me along this path.

Specifically, I would like to thank my co-authors, Nirnimesh and Parikshit Sakurikar, without whom this work could not have been completed. I am grateful for their valuable inputs and also for the various discussions that led to the practical systems presented in this thesis. Needless to say, without their inputs many of the topics touched in this work would not have seen the light of day.

I would also like to thank Jyotirmoy Banerjee, Yokesh Kumar, Anand Kumar, Jagdeesh Gorla, Bhavani Shankar, Akash Agarwal, Praneeth Shishtla, Sanjeev Pragda, Praveen Dasgi, Shashank Mittal, Prashant Pai and Vibhav Vineet for numerous discussions I had with them regarding various topics presented in this thesis. Lastly, I would also like to thank my friends and family members for helping me when I got in a rut and keeping my moral high while completing this work.

My acknowledgments also go to Microsoft Research, India for providing funding to present parts of this work at various conferences. I would also like to acknowledge Nvidia for providing the hardware for building the systems given as part of this work.

Abstract

Displays have seen many improvements over the years. With advancements in pixel resolution, color gamut, vertical refresh rates and power consumption, displays today have become more personal and accessible devices for sharing and feeding information. Advancements in computer human interaction have provided novel interactions with current displays. Touch panels are now common and provide better interaction with the cyberworld shown on a device. Displays today have many shortcomings still. These include a rectangular shape, low color gamut, low dynamic range, lack of simultaneous focus and context in a scene, lack of 3D viewing, etc. Efforts are being made to create better and more natural displays than 2D flat rectangular screens we see today. Much research has gone into designing better displays including 3D displays, focus and context displays, HDR displays, etc. Such displays enhance a display directly using device level technologies such as physical, chemical and metallurgical means. This approach has been taken in the past but proves expensive and is hard to scale to better standards that may be needed in color, refresh rate, spatial and intensity resolution in the coming future. New paradigms must be explored to generate better displays that may provide better user experiences and are easy to view and interact with. The conventional direct approach requires new physical properties to be constantly discovered in order to push the envelop in display design. Such technologies are hard to come by and may take years to prove their merit, for them to be acceptable in the display design pipeline.

An alternate is to use computation to enhance displays. Today computation is easily available and cheap. CPUs follow the Moore's law for their ever expanding compute capabilities. Multi-core CPU architectures are now common - even in hand held devices. GPUs consisting of 1500 cores deliver 1.5Tera FLOPS for $500 today. This abundance of compute capability has been applied to various systems with much success. Many areas including computational photography, computational biology and human computer interactions take advantage of computation to enhance a base system. Displays too have been shown to benefit from such an approach. Fish tank virtual reality is an example of this, 3D viewing can be enabled for a head tracked viewer using standard displays in conjunction with computer graphics. Computation can thus be used to enhance displays or remove their shortcomings. Many displays are underway that combine computation with other means to provide better and natural user experiences. These may combine computation with optics, display arrangement, metallurgy, sensors, etc. to bring out more than what is available on existing displays. Such displays can collectively fall under the term *Computational Displays*. There are two aspects to enhance displays using computation:

using hardware modifications and/or using algorithmic design. Computational displays, as a subject, covers both these grounds. While hardware modifications can range from a simple arrangement of display elements to designing a completely new hardware unit for a display system, the cost factor must be considered along with display applicability. Algorithmic design outshines hardware modification in this regard as it allows the use of any hardware and holds no bound on the design. Due to easy availability of computational devices much work can be ported to computational elements with limited dependence on hardware modification. We follow this line of thought in this thesis.

We propose computational displays which employ computation to economically alleviate some of the shortcomings of today's displays using algorithmic modification with limited use of hardware modification. We demonstrate the idea using four prototypes, which collectively demonstrate how computation can be used to enhance displays. Specifically we propose solutions to the 3D viewing, the focus+context and the limited color resolution problems in this thesis. We create four systems to this end, two dealing with rendering view-dependent 3D scenes to piecewise planar and non-planar surfaces, another giving a framework to render massive environments interactively to a tiled display wall, achieving focus and context for the scene, and the last providing better intensity resolution using three mixing methods. The systems work on top of existing display methodologies and are independent of any specific display technology. This makes our systems scalable to any displaying method and enhances the same using computation. In conclusion, we argue that computation/algorithms should be built into the displays themselves to enhance the visual experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Displays today are indispensable devices to view, share and feed information. They are essential part of our interaction with the virtual world and provide the only means of visualizing it. Nearly every device houses a display today ranging from LCD, OLED, E-ink on portable to CRT, PDP, DLP at large scales. Evolution of displays has taken interesting turns in the past decades. They have evolved from primitive black and white screens to the modern 16 million color OLED panels in a short amount of time. This has enhanced much in displays: color gamut, pixel resolution, vertical refresh, power consumption, contrast etc. The enhancements have provided us with much clearer, fluid and life like images. Interaction has also evolved with such displays. Computer human interaction (CHI) continually enables new ways of manipulating virtual content shown on displays. Touch panels are now common and can be found on nearly every device housing a display, from a microwave oven to elevators to personal music players. They have given us a way to interact with the virtual world shown on a display. These devices are so common that we rarely stop and think about them. We have accepted them as part of our environment, even so the virtual world they project is no longer unfamiliar.

Though displays have come a long way from their infancy, some of the fundamentals have not changed. A display is still conceived as a flat screen with which to interact with in two dimensions. This is far from how we view and interact with the real world in three dimensions. The initial concept of a display being a flat 2D screen has persisted amidst all advancements in display technology. We rarely think about displays in any other way. Very few examples of non-planar displays exist, even fewer of the 3D display. Display fundamentals must change in order to provide better and more natural experiences than what can be achieved by current methods. These fundamentals, in many ways, are responsible for many of the problems faced by displays today. Planarity of a display, for example, has hindered in providing natural interactions with a display, and even more so in enabling three dimensional viewing. On the color front, though the improvements have been there, displays still lag behind cameras in terms of intensity resolution. They also do not have high refresh rates and the pixel resolution is far less than what we had on an analog film. Attempts have been made to address some of these issues, both in the research community and more recently in the mainstream market, with companies taking up

the challenge to provide better visual experiences. Many shortcomings, however, still remain and have generated a series of fundamental problems that permeate displays today.

1.1 Problems Faced by Displays Today

There are many shortcomings of displays today. These include, inherent planarity, flat rectangular viewing region, limited focus and context, need for better interaction, need for user feedback, need for 3D viewing, better color gamut, higher dynamic range, better refresh rates, etc. The spectrum of problems is huge. At the core of these are few fundamental flaws in display design. These important aspects of a display system must be tackled as they spawn the various problems discussed above. In this thesis we cover only a small sub-set of these. In the following sections we describe each briefly.

1.1.1 Low Intensity Resolution

Color is an integral part of a display system and has been the focus of much research in recent years. The subset of colors reproduced by a device out of the full visible spectrum of light is called its gamut. Most devices have limited gamut and are not able to handle the intensity, saturation and resolution required to reproduce color accurately. Cameras today are capable of capturing 12-bits per channel. However, displays to view this information are not easily available as most displays are limited to a low color resolution of 8-bits per channel. The lack of high color resolution poses the high dynamic range (HDR) problem. A high dynamic range image cannot be displayed on current displays. This is due to the fact that HDR data gets decimated to the display intensity resolution, losing critical information that is required by many applications including scientific visualization of medical and astronomical data. Much research is going on in the area of high dynamic range imaging with many technologies promising higher color range in the near future. HDR displays to show such images are still far from being commonplace. Few attempts are now underway to tackle this shortcoming. Dedicated companies like BrightSide™ are now coming up with true HDR solutions [1]. However, they are still not mainstream. A handful of solutions exist today to render HDR images to low-bit displays. Most of these are image enhancement techniques employing tone mapping operators. Images rendered using these methods lose information in the process, though they are visually appealing [61, 60].

1.1.2 Displaying and Rendering More Pixels

More and more pixels are required to provide better detail to the content shown on devices. Rendering more pixels is a challenging problem even in the current age of fast GPUs. Displaying large number of pixels is also a hard problem. Cost of a display increases prohibitively with number of pixels and the density per unit area or resolution. This lack of high resolution has caused the focus+context dilemma. There is a trade off between resolution and display size in computer displays. The resolution of the display affects the visible detail and the size affects the visual context. Zooming in to view the fine

details results in a loss of the bigger picture. On current displays with limited resolution, one can either view the entire scene (context) or zoom into a portion of the display to view detail (focus). Both cannot be achieved simultaneously on current displays. Increasing the number of pixels or the resolution can address this shortcoming. Analog displays such as CRTs had the capability to increase resolution due to the continuous sweep of electron ray on a contiguous span of phosphors. They were however limited by the sweep speed. Today LCDs have fixed number of pixels, though on a growing trend, they are still far behind CRT displays. Analog displays too could not match the quality of the analog film, which was capable of capturing data at incredibly small scale. Focus and context is hard to achieve simultaneously on any display due to limited spatial resolution. Many attempts however, are underway to tackle this limitation either by increasing the spatial resolution [10, 3] or using novel methods of visualization [18]. Work is also going on in increasing selective parts of a projected image to give a perception of better image resolution. Sajadi et al. [95] prototyped an edge enhanced projector using an array of lenses in the projector's optical path. They also present a way to generate a close match to a high resolution input image using a low resolution projected image shifted and overlaid with itself in [96]. Wobulation based methods [16] have also shown to enhance display resolution by temporally mixing pixel intensity using motion or vibration of the display panel [22].

1.1.3 Low Refresh Rate

Lower refresh than that of 10 to 12Hz is perceived by the human visual system as individual images. Higher speed of screen refresh directly translates to a more fluid visual experience. For images displayed on a display to be life-like, high refresh rate is needed. Though the human visual system has the property that frame rates beyond 30Hz is indistinguishable by the eye, people (especially gamers) have suggested otherwise. A person playing a game at 120Hz refresh, feels the stutter if forced to play at 60Hz. Furthermore, with the advent of stereoscopic technologies to mainstream displays, high refresh rate becomes a necessary requirement. Display manufacturers have thus focused attention on this fact in recent years. Maximum refresh rate of common displays today is 240Hz, based on the blue-phase technology [48], which is limited by the physical materials used to construct the display.

1.1.4 Planarity

Displays today are flat, inactive rectangular standalone windows feeding images to an observer sitting in front of it. A reason for this model is the assumption that the data observed by the viewer lies on a planar surface. This approximation of the viewing mechanism, however, is far from how we observe our world in three dimensions. The gap in human perception and the display model has created a passive platform with problems of the likes of non-intuitive interaction and lack of 3D viewing amongst others. Few attempts have been made to create non-planar displays that show either 3D content or content suited for a particular shape, such as a globe on a spherical display [59]. Interaction has also been explored with non-planar displays [21]. New computer human interaction paradigms explore ways to make display

a part of the user space. This necessitates displays be made in form factors and shapes that are more natural to view and interact with. With the advent of flexible displays this can become a reality.

1.1.5 Lack of 3D viewing

This is by far the most obvious shortcoming of displays today. Everything we see in the real world is three dimensional. However, (despite the fact that displays have been around for nearly 80 years) not much has been done in this regard. Three dimensional display have been made. However, due to either lack of technology or lack of economical feasibility these have not made into the mainstream market. Content generation has also been a factor towards this. Three dimensional content generation is a research subject in itself. With the advent of new motion capture devices, depth per pixel can now be captured, making it easier to capture and generate 3D content [81].

Three dimensional displays today have become synonymous with stereoscopy due to aggressive marketing of this technology by leading display manufacturers. However, stereoscopic displays are only one of the ways to generate a 3D effect, and it is not a particularly accurate one. Stereoscopic displays show 3D only to a single viewer. The effect is generated by showing a depth offset image to the left and right eyes either simultaneously using polarization (passive stereo) or at high speed using shutter glasses (active stereo). The position of the viewer must be in a *sweet spot* to perceive the 3D effect, everyone else sees a distorted image. The sweet spot is usually fixed for such displays and image is displayed on a rectangular plane.

A better approach to perceive 3D is fish tank virtual reality (FTVR), which creates a volume of space around which a person can move around, view and interact with from any angle. The display is usually created as a cube which can be viewed from inside-out or outside-in configurations. The viewer's head is tracked in real time and view-dependent images are displayed on each of the planar facets of the cube to create a volumetric effect. The system also typically employs stereoscopy to generate left and right eye views. FTVR is usually limited to a single viewer, however, can be extended to multiple viewers if high speed displays and shutter glasses are used [62].

True 3D can only be achieved when everyone can view a virtual object from any angle. These displays do not pose any restriction on the viewer or the content, as they generate a pixel in space for everyone to see. This requires technologies such as laser scatter, auto-stereoscopy, mechanical rotating devices, etc. Though such displays are capable of producing true 3D effect, most of them are severely limited by the technological infancy of these methods. For example, laser scatter displays cannot show color and auto-stereoscopy can produce sudden shifts in the observed image with changing viewing angles. Sony recently showcased the raymodeler [111], a 360°walkaround mechanical 3D display. It and many other such prototypes are limited in size due to their mechanical design. They are hard to scale to larger sizes and are limited in resolution. Though in their infancy, these technologies have the potential to solve on of the most challenging problems in display design.

1.1.6 Lack of Interaction

Interaction with displays has improved over the past few years, from mouse, joysticks to touch screens and now full body motion capture. All of which are used to move a selector in two dimensions. A few 3D mice and motion capture devices are available, but are not mainstream yet. Interaction with non-planar surfaces and projections on to any surface have also been explored [51, 21]. Virtual interaction with 3D content on both FTVR and holographic displays is also underway [20, 111]. Displays, however, are still limited in feedback given to the user. Most of these include force feedback systems ranging from steering wheels to complex haptic contraptions. Interaction with a conventional display is limited by many aspects of the display system, chief amongst which is its inability to display 3D. This hinders a user as he/she cannot estimate the depth of a virtual object correctly. This limits the interaction to a 2D plane. Interaction too is limited by the visualization method used. Together these have created the lack of rich interactions with a display. The subject of interaction is highly intertwined with display design, as the human interaction factors must be considered to be at the core of the display design to enable natural interaction. For example, interacting with a 3D display requires a person to reach into the 3D space to virtually touch the object. This requires the display to accommodate such interaction, rotating mechanical devices, for example, cannot accommodate this. Much research is going on in building better methods for interacting with displays. We should also simultaneously look at the fundamental design flaws in a display system. Such an approach would be much more robust than trying to pack content on to flat screens.

1.2 Market Solutions and Future Trends

Much research has gone into dealing with the above limitations and many have been met with success. We will discuss these in the related work sections of independent solutions we propose later in the thesis. Here we present the displays and prototypes that are already present or are upcoming in the mainstream market which attempt to solve the aforementioned problems.

Mainstream market has taken the direct technical route for tackling with the reported display related problems, relying on the conventional methods such as chemical, physical, metallurgical etc. to enhance system capabilities. This route is comparable to a brute force approach in which only technical limits of materials and their properties are exploited in order to gain system advantage. For example, to compensate for the spatial resolution, size of the display has increased from 14 to 150 inches with a maximum resolution of 4K pixels (Figure 1.1). Resolution too has been packed into smaller form factors such as computer monitors and more recently hand held displays. This provides better focus and context as the image quality is retained. The technology, however, remains the same, LCD or OLED. Smaller and better packing mechanisms enable high density packing of pixels to create true 4K resolutions. JVC projectors display 4K resolution using E-shift [5], which creates two images of 2K resolution offset by half a pixel in both dimensions to create double the resolution. The method produces sharper details but is a simulated approach and cannot display true 4K input content.

Figure 1.1 The upcoming 4K resolution projector and display from Sony and LG

Special displays have also been showcased to display HDR images, which work at the pixel level to block inter pixel leakage to create an HDR effect. The focus here is to create high contrast ratio, which can then be quantized into high bit color intensities. Figure 1.2 shows a display created by BrightSide compared to a conventional display. The image clearly looks superior to that on the conventional display. Conventional display light pixels using a single light source behind the LCD screen spread across the entire screen using a diffuser. Based on the location and the type of the light source, various product nomenclatures are assigned. For example, LED-LCD uses LEDs as a source light as opposed to fluorescent tubes. These can further be arranged in edge, border or per-pixel manner. The BrightSide display uses 1400 LEDs and an LCD panel along with image enhancement algorithms to create a practical contrast ratio of $2 \times 10^5 : 1$ [1].

Using the direct approach to produce high refresh rates has been met with some success. The fuel for this comes from the need to display stereoscopic content, termed ¨3D˝ in market nomenclature. The term, 3D, however, is a false description as the display produces correct 3D only from a fixed view point. More and more companies are providing stereoscopic display solutions. Most use active stereoscopy. This requires image to be shown twice, one from the point of view of each eye while shutting off the image from the other using active shutter glasses. This doubles the frame rate required in the monoscopic mode. According to the human visual system, on smaller screens (up to 15 inches) a 60Hz lowest refresh rate produces a ¨smooth˝ visual experience. However this must increase to 72Hz for larger form factors - 17 inches and above [6]. Most displays are, however, limited to 60Hz refresh rate due to technological and economical constraints. For a stereoscopic display, however, this limit must be doubled, i.e. at least 120Hz is needed, which are hard to achieve with high LCD response times. Hacks such as producing intermediate images and interleaving them with frames are typically

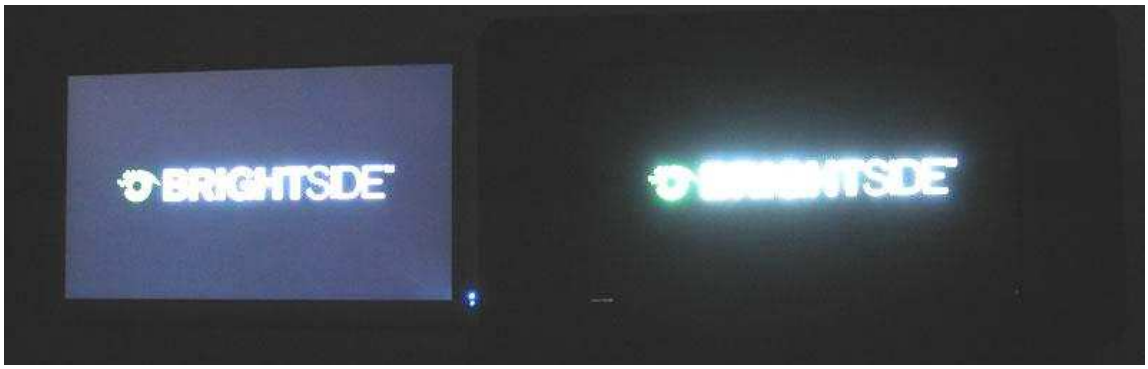


Figure 1.2 The BrightSide prototype HDR display (on the right) compared to a high contrast conventional display (left).

used to increase the refresh rate, these methods can produce high refresh rates of up to 600Hz, but are again hindered by the LCD response times. For stereoscopic displays, however, such hacks are useless as a true high refresh is needed in order to produce a smooth image in the eye of a viewer. New technologies are thus needed, such as the Samsung Blue Phase [48], which employs material properties to reproduce true 240Hz displays.

Though sported as 3D displays, stereoscopic displays are not truly 3D as mentioned. True 3D displays provide 3D viewing from all angles. Multiple variations of these exist in the research community: ranging from fish tank virtual reality displays to true holographic displays. No mainstream product, however, exists as of now. Much work is going on in auto-stereoscopic displays. Wetzstein et. al recently prototyped Tensor displays [123], which produce auto-stereoscopic effect using multiple LCD panels by factorizing the image for each LCD panel for light field data. Auto stereoscopic displays produce *viewing zones*, in which a viewer sees the correct left and right eye views [41]. Moving across these zones can produce the inverse stereoscopic effect - when the left eye views the right image and vice versa. This problem cannot be compensated as the zones are fixed in space. Further auto-stereoscopic displays can produce a ghosting effect while moving away from the prescribed zones, in which image from multiple zones overlap to create the viewer image. Sony recently showcased its holographic 3D display, called the RayModeler [111], based on mechanically rotating an LED array (Figure 1.3). The display is capable of true 3D reproduction from any angle. However, due to its mechanical design is limited in size and resolution and also suffers from the above mentioned auto-stereoscopic artifacts. It, however, is a good prototype of a technology that may become mainstream in a decade or two.

With a few exceptions, planarity of a display has not been challenged. With the need of natural interaction one must consider arbitrary shaped displays. Such displays can fit ergonomically well within the user space and provide natural ways to view and interact with data. Microsoft surface [75] and sphere [21] are initial attempts in this direction along with displaying and interacting with on any surface [51]. Interaction may also improve from today's touch panels and full body motion capture to gen-

Figure 1.3 The Sony RayModeler prototype: holographic 3D display based on mechanical LED rotation.

uine force feedback systems. These goals are, however, far fetched with tangents arcing over decades. Though the direct approach will provide solutions in the long run, one should also look at alternate methods to solve display related problems.

1.3 The Economical Considerations

Another aspect to consider while tackling these problems is the cost factor. Although the market's direct approach can potentially solve all display related problems given a long enough period of time, the cost of designing a practical system is too high at any given time. The upcoming 4K displays cost thousands of dollars and specialized prototypes such as the Sony raymodeler can go in the vicinity of hundreds of thousands of dollars. An economical approach is thus needed to produce viable products for everyone to use. The technology too limits what can be achieved at any given moment for the direct technical approach. For example, though an improvement over existing displays, why is the spatial resolution limited to 4K for current displays? Or why can't the refresh go beyond 240Hz? These are some of the questions that faces the direct technical approach today. Direct methods can only go as far as the technology and feasibility allows them. Scaling beyond these barriers is impossible with this approach. Intelligent use of display technology is thus needed to resolve display related problems without depending too much on physical, chemical and metallurgical means.

1.4 Computational Displays

Computation has proven a good ally to get more out of a base system than the direct approach relying on physical, optical or chemical means. Computation, too, cannot be restricted in ways other methods are limited within their physical boundaries. Computation today is cheap and abundantly available. CPUs follow the Moore's law with an ever increasing multi-core architecture, a graphics card costing $\$500$ today can touch 1 Tera Flops of computing power. Using computation to enhance systems has been tried with success in many domains such as computational photography, computational biology, etc. The concept is to get more out of a base system using algorithmic modifications. In computational photography, for example, a standard camera can be used to take an HDR image [37], or to take an all-in-focus image [63], etc. This is facilitated by the knowledge of the underlying working of the camera. This prior knowledge can then be used to enhance the camera to do more than what is available on a base camera. We extend the notion of algorithmic modification to displays and propose to enhance displays using computation. We call such displays *Computational Displays*. These displays enhance capabilities of current displays by careful use of computation. Computation may be embedded into each display or supplied externally to coordinate multiple displays together. The concept is described in Section 1.9.

The idea of computational displays has gained some attention in recent years. A course on computational displays presented at SIGGRAPH 2012 talks about various computational displays from modified projector optics to producing specialized prototypes capable of generating auto-stereoscopy [122]. The topics covered in this course essentially follow the same design methodology we present in this thesis. The course covers displays that are designed better using computation, based on which hardware modifications are applied to produce the desired result. Such prototypes all fall under the ambit of computational displays, however, have a distinct focus on hardware modification. The resulting image is thus no longer displayed on a conventional display but on a specialized display. This sets these prototypes apart from our approach presented in this thesis. We focus on algorithmic modification of the input image which is shown directly using conventional methods - thus is more widely applicable than specialized computational displays. This is a smaller yet important niche in the landscape that covers computational displays. Our approach treats conventional displays as black boxes, with more focus on inter-display processes than intra-display processes. Using these fundamentals, our approach becomes independent of the underlying hardware and can hence be used directly on even better hardware providing prolonged scalability. Content being shown is also standard, enabling our approach to easily use existing content. The main advantage of this approach is, however, the cost factor. By using algorithmic modifications cost can be reduced dramatically as shown by our prototypes. All hardware used in our systems is off-the-shelf with no hardware modifications. Our approach can thus be categorized as computationally enhanced displays - a subcategory of the larger computational displays framework. In the spirit of this thesis, however, we treat our approach as computational displays and henceforth refer to it as the same.

1.5 Hypothesis

The hypothesis we propose in this work is that computation, if used intelligently along with other methods, can compensate for many of the display shortcomings. This extends many existing ideas and prototypes. The use of computation to resolve display related problems is not a new one. Many attempts in this spirit have been made in the past. The idea may be traced back to head tracked based virtual reality displays for single [38] and multiple planes [35]. Most notable are FTVR displays which use head tracking with standard projectors to create an immersive 3D environment for virtual object viewing and manipulation. Though such displays were not known as computational displays, they fall under a common ambit. Another example is haptics, that create touch and feel for virtual object interaction and manipulation on a display. Such existing prototypes argue in favor of the idea that computation should be built into the devices themselves for enhanced viewing and interaction.

1.6 Related Domains

Using computation to enhance systems has been tried and tested in many other areas. The approach is similar to what we propose - intelligent use of computation along with prior knowledge of the physical processes involved in the system. A good example of this is *Computational Photography*, where camera/sensors are enhanced to capture more than just traditional imagery. The subject can be further divided into multiple disciplines such as computational illumination, computational optics, computational processing and computational sensors. The goal, however, remains the same, to gain more that what is available on the base camera by processing or controlling the image based on prior knowledge of the physical processes. Cameras are traditionally limited to a single viewpoint, focus and image plane. Numerous cameras have overcome these and other limitations using computation. For example, in the work given in [63], Kuthirummal et al. move the camera sensor and take images at various planes that is later processed to produce an all-in-focus image of the captured scene. Debevec et al. [37] show how to take multiple images at different exposures and accumulating the data to produce a high dynamic range representation of the scene. The omni directional camera by Nayar et al. [79], falls under the computational optics and can take a 360° view using a catadioptric mirror along with epipolar geometry. Raskar et al. prototype a non photo realistic camera that takes images of physical scenes using multiple flashes which can later be processed to produce any NPR effect [89]. In [117] Veeraraghavan et al. produce a light field camera using an aperture pattern set between the optical path of the camera. The camera is capable of capturing the light field entering the camera lens and can produce views from different viewpoints or change the focus plane later using computation. Levoy et al. produce a light field camera using mechanical gantry to capture a single slab of light field in [68]. Later they also show ways to capture any plane in partially occluded environments based on confocal imaging using an array of cameras and projectors [67].

As stated before computational displays too are attracting much attention in recent times. Many prototypes have been showcased to highlight what can be done with computation along with hardware modifications. Such displays include tensor displays by Wetzstein et al [123] which spawn a family of auto-stereoscopic displays designed using multiple LCDs/directional back-lighting inspired by the slit barrier/lenticular design [71]. The displays are displayed upon using a non-negative tensor factorization (NTF) of the input light field which generates independent patterns on each LCD to produce an auto-stereoscopic effect. The work extends over their previous work in which NTF is applied to two LCDs panels to produce a similar effect [64]. Sajadi et al also follow the computational display design methodology to produce n times better image quality for a given projector using pixel overlay and shift of the projector image with itself using lens arrays [96]. They map the problem onto a set of over constrained system of linear equations and provide a real-time GPU solver for the same. The final result is a good approximation of the high resolution input image. They also prototype a content adaptive resolution enhancement method by modifying the optical path of a projector using multiple LCD panels sandwiching an optical sharing unit [95]. The problem of image enhancement can then be mapped to a graph which can be optimized to produce required images on the different LCD panels according to the input content. Berthouzoz et al extend the wobulation method [16] method to LCD panels using a motor to shift the panel at high speeds in both directions, thus enhancing the perceived resolution in the observer's eye [22]. Such prototypes along with computational photography all argue in favor of computation as a necessary tool for enhancing a basic system. The idea can be extended to any system. We explore the notion in terms of enhancing displaying capabilities of conventional displays.

1.7 Scope of the work

In this work, we focus on creating displays that are readily usable with current available methods of content generation, rendering and displaying techniques. The systems proposed in this thesis use standard hardware along with computation to enhance displays. Our systems are easy to setup, maintain and calibrate, making them practical and usable. The systems are capable of displaying standard content with low or no modification, making such displays accessible to a large number of applications. We focus on inter display processes - where multiple conventional displays are used to gain more out of a system, intra-display processes without hardware modification is also explored. We treat conventional displays as black boxes, thus decoupling the systems from hardware dependencies. This type of arrangement allows greater scalability to computational displays. Our methods may be applied to existing displays with little or no hardware modifications. As stated, this covers limited ground in the landscape of computational displays - but is crucial component which highlights the use of computation in display design. To this end, we propose four software/hardware systems that resolve three of the display related shortcomings.

1.8 Thesis Statement and Goals

In this thesis we tackle three display shortcomings by proposing solutions to: (a) Correct viewing of 3D content on displays of arbitrary shapes, (b) improving color resolution of current displays for displaying high dynamic range content, and (c) tackling focus+context using tiled displays. This set of problems hardly scratch the surface of what can be achieved in the field of computational display. They are, however, chosen for their utility in common applications and also for their research merit. The problems are also chosen based on the physical and computational processes involved with them. Further, the problem set is orthogonal and unrelated. They thus explore different aspects of the fundamental display problems. Orthogonality of these research problems ensure that we are explore individual areas using the same hypothesis - that is the same hypothesis can be used to solve a wide range of problems. And the solutions we provide to these problems validates our hypothesis by example.

1.9 Computational Display Design Overview

As stated, computation can help enhance displays, the idea is to modify a display by applying careful use of computation. This can be done at many levels. In the design process, one can use computation to resolve many design flaws, in the manufacturing process computation can be used to enhance system capabilities. Computation can also work at deeper levels, such as, at each of the physical processes involved in the system - from metallurgical, optical, electrical to physical arrangement of pixels. We define *computational displays* as those in which computation is used to enhance displays at the algorithmic modification level. This may involve using external agencies such as optics, electronics, factorization etc., in conjunction with existing display technologies. This differs from the direct technical approach with applied computation, in which technologies are enhanced to produce better displays. The computational displays approach enhances displays with algorithmic modification applied to pre-compute images in accordance to an applied hardware modification. Please see Section 1.6 for examples.

The approach taken in this thesis slightly differs from the prototypes presented in Section 1.6 that significantly modify display hardware with physical entities. This is because we use conventional displays to gain more out of them using computation alone. We thus map our problems to existing display technologies rather than generating specialized hardware for the same. Figure 1.4 gives an overview of our approach to computational displays. Multiple standard inputs are taken to the system to provide a wide application base. The inputs to the design are (a) user inputs taken from standard input devices to interact with content being displayed and also specific inputs taken from specialized devices that may provide system specific data, (b) standard content is shown on to conventional displays to ensures that a wide range of applications can be ported to our design, and (c) prior knowledge: Our design allows for inter and intra display processes. For the inter-display processes, we input the prior knowledge of physical arrangement of display elements into the system. For the intra-display processes this knowledge corresponds to the various models that describe the display along with other physical processes that may

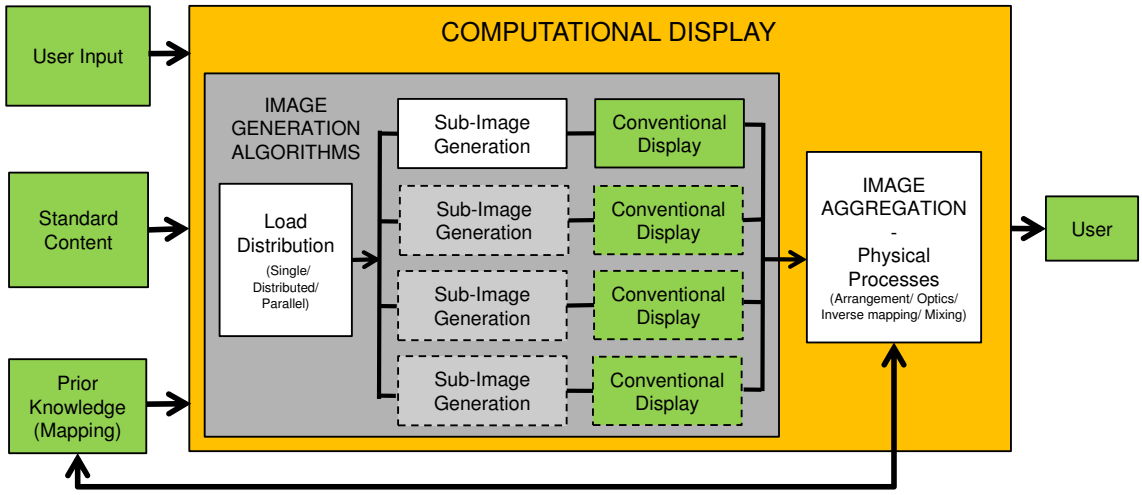


Figure 1.4 The computational display framework and the processes involved.

affect the perception of the final image. This knowledge is then used to create sub-images through various load distribution techniques which may follow a single, parallel or distributed algorithmic design. Primarily, the focus lies on image generation techniques, with prior knowledge of the physical processes involved in the system after rendering. We do not modify the display hardware, thus one or a series of conventional displays are used to produce the final image. The sub-images are shown on conventional displays arranged in accordance to the inter/intra display processes to produce the enhanced display. For example, for a tiled display setup each conventional display shows a sub-image which is a part of a larger screen, with pre determined location within the context of the larger screen. This knowledge is then used to render the part of the screen to that particular tile, generating a high resolution image collectively. The arrangement of individual images and their mixing are physical processes that may be mechanical, optical or electrical in nature. The physical process involved in the system are known, in case of a tiled display a simple tiling of the sub-image in the larger image specifies this. In other examples these can be more involved .

In our design, algorithmic modification can apply to a single or distributed or parallel channels for generating the desired image. The load distribution is problem-dependent. As shown in Figure 1.4 the framework allows for any number of ways of approach. In the case of a tiled display, each tile can render its own part independently using distributed or parallel rendering, or can simply show an image given to it by a centralized machine - thus allowing the options to choose a client-server or a master-slave framework for its implementation. The end result is presented to the user using conventional display methods, which results in an enhanced display experience to the viewer. We do not opt for hardware display modifications for two reasons: (a) We want to create a framework that can be used and crafted using simple off-the-shelf components such that it can be applied on to any given display technology. (b) We aim to show generic content. Content creation for specialized hardware is a difficult undertaking

that can restrict the application of a display system. We seek to provide with a framework that can provide better visual experiences at low cost in this thesis, and Figure 1.4 epitomizes this approach clearly. Later in each of our prototypes we show how this framework applies to individual problems and how algorithmic modification along with off-the-shelf hardware can be used to provide better visual experiences not available on current displays.

1.10 Contributions of the Thesis

This thesis covers three fundamental problems relating to displays - each resolved by designing a practical and usable system based on the computational display framework. The focus+context problem is tackled by tiling conventional displays and off-the-shelf hardware using distributed rendering. Intensity resolution of a display is increased using three novel mixing methods. Viewing 3D on to arbitrary shapes is dealt with in two ways: using multi-planar or non-planar surfaces. Because the two approaches are independent, we propose two separate systems for this problem.

The research presented in this thesis provides novel ways to design and built displays, and provides an outlook on how to approach display related problems using computation. The main contributions of this thesis are (a) resolving three fundamental display problems, (b) extending research in each of these areas by providing novel and scalable solutions using off-the-shelf hardware, and (c) concertizing the idea that computation should be part of display design as much as mechanical, optical or electrical means, by example. We give a short overview of the systems presented in this thesis and on what grounds they contribute to the research in the respective domains.

1.10.1 Perspectively Correct Multi-Planar Displays

Displays have remained flat and passive amidst the many changes in their fundamental technologies. One natural step ahead is to create displays that merge seamlessly in shape and appearance with one's natural surroundings. In this work, we present a system to design, render to, and build view-dependent *multi-planar displays* of arbitrary piecewise-planar shapes, built using polygonal facets. Our system is designed to provide a high quality, interactive rendering of 3D environments to a head-tracked viewer on arbitrary multi-planar displays. The system is designed to render to large number of display facets by rendering all facets in a single pass of rasterization using a novel artifact-free rendering scheme. A parallel, per-frame, view-dependent binning and pre-warping of scene triangles adds scalability to the system, allowing interactive rendering of complex scenes to arbitrary complex display shapes. The method places no constraints on the scene or the display and allows for fully dynamic scenes to be rendered interactively at high resolutions, producing accurate artifact-free visualizations on to any given surface. The steps of our system are implemented efficiently on commodity GPUs, keeping the cost low and allowing the system to exploit data-parallel hardware. We demonstrate the use of our system using a few real and simulated prototype displays of different shapes, form factors, and complexity: from

a cube made out of LCD panels to spherical/cylindrical projected setups to arbitrary complex shapes in simulation. Performance of our system is evaluated both in terms of rendering quality and speed, for increasing scene complexity and display facet sizes. A subjective user study is also presented to evaluate the user experience using a walk-around spherical display as compared to a flat panel in a game-like setting.

1.10.2 View Dependent Rendering to Parametric Displays

In this research we explore displays whose surfaces could be defined by a set of parametric or implicit equations, however, with a fixed mapping to a 2D domain for each pixel. The mapping is input to the system and must be considered while designing the display. Most common algebraic shapes can be embedded onto planes, more complex shapes can be segregated into a number of smaller forms that map onto a 2D plane. A display surface designed in this way could have a number of arbitrary curved shapes given by implicit or parametric equations with a mapping to a 2D plane. We present a fast and efficient method to render 3D scenes onto such a display in a perspectively correct manner to a user specifiable error bound. Our method renders the scene by tessellating it based on the geodesic edge length up to the user-defined error. Decreasing the error produces better results, however, reduces interactivity. User can adjust the error to what is suitable for him given the parametric surface - our results show decreasing the error beyond a certain point does not increase quality. Tessellating the scene alone is not good enough to produce a quality rendering of the scene, the rendering must also compensate for the display curvature. In order to do this we modify the scene vertices using per-vertex ray casting, such that the final image appears correct to the user's viewpoint. The ray-surface intersection procedure, geodesic length computation and 2D image mapping are assumed to be known for the given surface. We exploit the tessellation hardware of the SM 5.0 GPUs to perform the error checking, polygon splitting, and rendering in a single rendering pass. This brings the performance of our approach closer to rasterization schemes, without needing ray tracing. Our scheme does not interpolate pixels, ensuring high quality renderings. We demonstrate a few real display prototypes and show the scalability of our system using simulated scenarios.

1.10.3 Intensity Mixing to Display HDR Images

Color intensity resolution has not changed much over the past few decades. Most displays are still limited to 8-bits per channel, while simultaneously much work is going on into capturing high dynamic range (HDR) images. Mapping these directly to current displays decimates the HDR data and necessarily loses information that may be critical to many applications. In this work, we present a way to enhance intensity resolution of a given display by mixing intensities over spatial or temporal domains. Our system sacrifices high vertical refresh and spatial resolution in order to gain intensity resolution. We increase the intensity resolution of a given display at lower refresh rates or spatial resolution while keeping the same contrast range of the display. The method thus increases granularity of the available

contrast. We present three ways to mix intensities: spatially, temporally and spatio-temporally. The systems produce in-between-intensities not present on the base display, which are clearly distinguishable by the naked eye. The better granularity of the intensity range enables a viewer to see the in-between-intensities not present on the base display and make distinctions based on them. We evaluate our systems using both a camera and human subjects, evaluating if our system scales the intensity resolution and also ensuring the newly generated intensities follow the expected display behavior.

1.10.4 The Garuda Display Wall

Garuda[1] is a client-server based display wall system employing distributed rendering to render massive 3D environments at interactive frame-rates to a tiled display. The system is designed and built using commodity hardware. Features such as client caching and use of server-push philosophy in conjunction with UDP multi-cast help the system scale to very large tiled configurations. The system uses a novel culling algorithm to find what part of the scene is in which of the tiles. This is a scalable algorithm both in terms of scene complexity and number of tiles. The system is built as a library intercept mechanism over the Open Scene Graph (OSG) API. This ensures that any OSG application can be ported to Garuda without the need of modifying, recompiling or relinking the code. The OSG executable directly runs on the tiled display using this. Rendering capabilities increase with number of tiles as distributed rendering is employed. No machine in Garuda, server or clients, renders the entire environment. Culling is performed at the server and part of the scene graph are sent to clients over standard Ethernet. Clients cache the geometry and evict it based on LRU, this exploits the temporal coherence in the scene. Rendering happens at client end with the server used to to synchronize the buffer swap.

1.11 Organization of the Thesis

Hereafter each chapter explains in detail the theory and implementation of each of our systems - which deal with one of the fundamental problems stated above. A total of four systems are presented dealing with three of the fundamental research problems stated above. The work collectively reinforces the notion of using computation to enhance base systems. Our systems are implemented using standard hardware, making them practical. Speed and accuracy are the primary concerns in any display system, and are at the core of our work, along with scalability. Our systems are independent of specific display technologies. The focus is on display enhancing algorithms and how they can be used to create better displays. A literature survey is due in context of a thesis. However, since our systems explore fundamental dimensions of display problems it makes more sense to explore the literature in individual chapters themselves. The systems all follow the computational display framework presented in this chapter - with more focus in algorithmic modification and less on hardware modifications. Each system

[1]This system is joint effort with another student. It is given here for completeness. Specifically, the design of the adaptive culling algorithm and detailed experimental analysis were my contributions.

is presented with introduction to its problem statement and how the system derives from the computational displays framework. The background and related work for each of the research problems are then discussed. The implementation details and algorithmic modification follow this in each chapter. Experimental evaluation along with display prototypes are presented thereafter, with conclusions and future work analyzing the system arcs.

Chapter 2 explains the multi-planar system and the theory behind the novel accurate rendering used in it. It also presents novel ideas on how to scale the system to very large display configurations. Chapter 3 presents a new way to render to displays that may be defined using parametric or implicit equations. Implementations of the scheme on spherical, cylindrical and paraboloid shapes are presented that establishes the system's capability to handle any form factor. Experimental evaluation presents practical speeds achieved on these setups in comparison to the multi-planar framework presented in Chapter 2. In Chapter 4, we present the system to display HDR images onto existing displays. We evaluate the system using both a camera and human subjects to make sure they follow the expected intensity curve. Lastly, in Chapter 5 we present the Garuda display wall. We show various tiled configurations alongwith experimental studies to evaluate scalability and bottlenecks of the system. Chapter 6 presents the conclusions derived from these systems and what may be the future direction of this research area.

Chapter 2

Designing Perspectively Correct Multi-Planar Displays

2.1 Introduction

Much work has gone into capturing and creating 3D content. A lot of three-dimensional content – ranging from games to CAD simulations – are viewed by users on computer screens today. New modes of real-time 3D content capture are now possible [81]. Virtual reality, haptics, and computer human interaction all deal with the central idea to seamlessly transition from the real to the virtual world. A necessary extension of this paradigm should include displays that seamlessly merge with the world and are natural to view and interact with. The goal of such a display is to blur the boundary between the real and the displayed objects. This notion has two parts: (a) the display itself being part of the viewer space and (b) content displayed on it being an accurate depiction of the underlying world. The first condition necessitates displays to approximate arbitrary user spaces. Rendered images on such a display must also be accurate such that the depiction agrees with the natural perspective of the viewer.

We present a framework to design, build, and render to *multi-planar displays*, built using multiple polygonal *facets*. We render perspectively correct 3D on to such a display from the viewpoint of a single head tracked viewer. Our framework effectively extends FTVR displays to generic polygonal shapes. Our system scales to arbitrary polygonal shapes, which sets it apart from other FTVR displays which are limited to a few facets. Rendering to FTVR displays has a trade-off between rendering quality and the number of facets. Projective Texture Mapping (PTM) can produce FTVR effect on any surface, but suffers from quality degradation [24]. Off-axis rendering can produce quality images at higher rendering costs but suffers from depth artifacts [14].

We present a novel rendering scheme that produces correct images on planar facet and across facet boundaries, generating a collective view of the virtual world. The rendering cost is reduced using a parallel binning of scene triangles to the display facets, implemented on commodity GPUs. We also render all facet images in a single pass of rasterization, generating one or more *quilt images*. The quilt image is un-mapped to the display geometry either using multiple VGA outputs or using specialized hardware for the same. Our system is implemented on commodity GPUs and can scale to displays consisting of well over a thousand facets. Our design provides both the quality and the interactivity needed in appli-

cations such as medical visualization, computer aided design, simulations and games. The framework contributes (a) a system to drive a generic multi-planar display through one or more rectangular quilt images which can be generated by a standard graphics system retaining pixel density, (b) a method to correctly render images to a set of arbitrarily oriented planar shapes to provide a perspectively correct image as well as depth across facets boundaries, and (c) a scalable, single-pass rendering mechanism that sorts and prewarps the scene triangles mapping to each facet, enabling rendering of all facets in a single pass of rasterization. Interactive rendering rates and correct facet rendering are the guiding concerns in designing our system. The system is capable of rendering fully dynamic, deformable, triangulated scenes containing over 200K triangles to a display shape consisting of over 1600 facets at 35Hz. The setup uses a single Nvidia GTX580 GPU for rendering and also for parallel sorting. This can be further improved with better hardware and the use of multiple GPUs.

2.2 Related Work

Our work is related to several previous efforts including multi-planar, non-planar, curved and FTVR displays. We review the literature related to these in this section, segregated into multi-surface displays, holographic displays, FTVR displays and rendering to FTVR displays.

Multisurface Displays Multiple display panels using monitors or projected surfaces have been used to increase pixel resolution for various applications [94]. Tiled display walls using LCDs or projectors scale this to extremely high resolutions [83, 107]. Display walls allow focus and context to be achieved simultaneously [49], which is also achieved using non-tiled arrangements such as displaying a high-resolution focus window within a lower resolution context [18]. Non planar displays have also been explored. Raskar et al. modify object appearance for any given geometry in [91], which is further enhanced using multiple projectors to create high-resolution appearance editing [65]. Such advancements have spawned much work in auto-calibrating systems for multi projector displays [97], both in terms of color and geometric corrections. Other non-planar displays that enable users to interact with information content from all angles have also been showcased [59, 104, 75]. Multisurface displays usually show information with multiple parts of the display showing different data. Single or multiple users can interact and share the same display using touch gestures [21]. Multisurface displays provide novel interaction and appearance onto arbitrary shapes, but are limited to two dimensional information content. The mutiplanar display we propose aims to show 3D content accurately in addition to 2D information.

Holographic and Volumetric Displays Three dimensional displays are being actively pursued by researchers today. A comprehensive review of various 3D image generation and displaying technologies is given in [110, 80]. Volumetric and holographic displays are true 3D displays capable of displaying imagery in space without eye-wear. Technologies such as laser scattering [124], plasma generation [13] have been demonstrated before. Jones et al. presented a novel multi-viewpoint 3D display using a high

speed projector and complex, synchronized rotation arrangement [58, 42]. The display could not show colors, had low vertical refresh rates and was small in size. Methods involving lenticular lenses and integral photography were used for auto-stereoscopic displays [70]. GCubik used integral photography to pre-synthesize images for each facet to display a 3D object from any angle without head tracking [72]. The display had low resolution and caused sudden shift in observed image when the viewing angle shifted. Technological requirements are excessive to make true 3D displays practical today. They, however, hold promise as future 3D displaying technologies.

FTVR Displays FTVR displays have extended multisurface displays to render volumetric data for one or more head-tracked viewers. An early immersive implementation is the CAVE virtual environment [35] that uses four back-lit projection planes enclosing the viewer. View-dependent images are displayed on the projection screens by tracking the viewer's head. Stereoscopic projection is used to further enhance the experience. Recent FTVR displays include Cubby with three back-lit projection screens and a head tracker for a small closed FTVR environment that allows interaction with virtual objects using precision tools [40, 44]. By inverting the facets of a CAVE, with the facets of the display pointing outwards, Cubee creates a walk-around cubic FTVR display [113]. PCubee enhances this to a hand-held display coupled with a motion sensor and a head tracker [112]. The display can then be moved around and observed from any angle. FTVR displays are limited to small number of facets, usually a cube, due to increase in rendering load as the number of facets increase. Iwata implemented a non-cubic, rhombic dodecahedral display using projectors to cover the full solid angle [56].

Rendering 3D Content To FTVR Displays Off-axis rendering is used to render 3D environments to FTVR displays. Deering proposed accurate head tracking along with stereo image pair generation for a single display [38]. This was extended to multiple planes in the CAVE virtual environment [35]. The method renders each facet using an off-axis, asymmetric frustum to cover the display rectangle. Rendering load increases linearly with the number of facets. The method, though extensively used, can produce geometrically incorrect images at the periphery of the frustum for an outside-to-inside display configuration as discussed in Section 2.5. Correct view can be produced using ray-casting at every pixel for any given display shape albeit at high costs. Hou et al. proposes a multi-perspective rendering method for any given surface [54]. Though not intended for FTVR displays, it can be extended and employed for the same. The method is implemented on GPUs using appropriate shaders and can handle dynamic scenes. It, however, interpolates barycentric coordinates per pixel and requires back projecting of rays at every pixel - which translates to per pixel raycasting and thus slows down the rendering speed for large scenes. Another alternative is to use homography pre-warping to render to FTVR displays [17, 88]. Our previous work uses this method to create a polyhedral walk-around display [50]. A scalable culling pipeline is required to feed FTVR rendering methods if arbitrary number of planes are to be used in a display configuration.

2.3 As a Computational Display

The Multi-Planar system we present maps directly to the computational display framework presented in Section 1.9. The system is designed using prior knowledge of the physical processes involved in the setup and generates images based on it using computation. The end result is presented to the user using known/existing display methods to produce a display system that does more than what can be achieved using conventional displays. The presentation is our case maps to an array of conventional displays arranged in a desired configuration. This directly maps to the computational display framework, as shown in Figure 2.1. The image generation follows parallel load distribution which results in best performance even when the number of tiles increase to over a thousand facets. We show standard content with user interaction on multi-planar displays which also agrees with the computational display framework. We briefly give an overview of the physical processes involved in the Multi-planar system and the algorithmic approach to compensate for them.

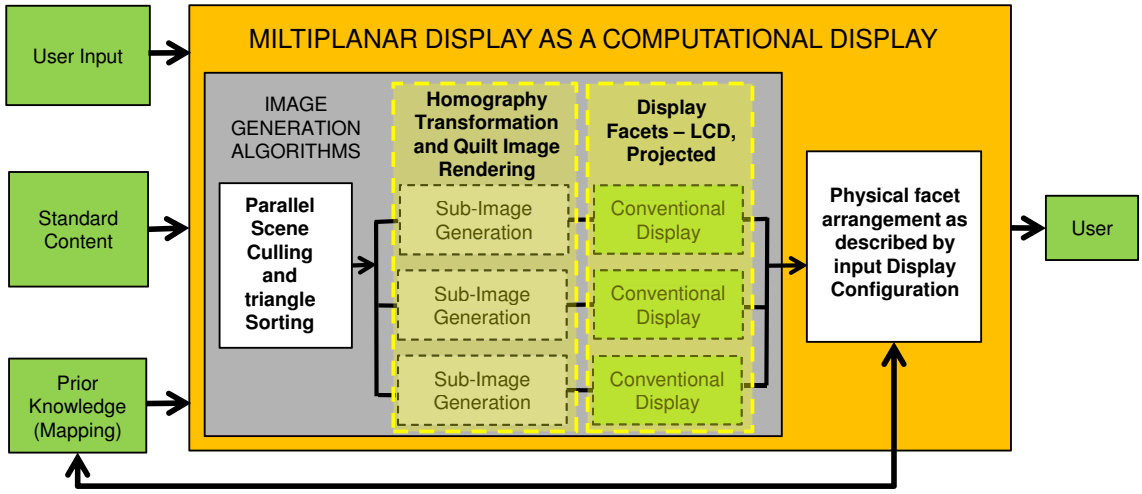


Figure 2.1 Multi-Planar Framework as a Computational Display

2.3.1 Physical Processes Involved

A number of physical processes are involved in a multi-planar display system. As stated, the main idea is to produce a 3D view in the observer's eye given an arbitrary multi-planar surface. Correct depth perception is thus a necessary requirement. This cannot be achieved using conventional displays because of their inherent planarity, as it restricts the viewable region visible to a observer. To generate the effect, we assume a surface built using multiple planar facets. The viewer's head position is tracked in a known coordinate frame. The display and its constituents are also calibrated in this coordinate system. Using

these information we generate images on individual display facets to produce a corrective image from the point of view of the observer. The correction of each facet plane with respect to the viewer location is the main compensation factor. We also extend this to a scalable culling of scene triangles for better performance.

2.3.2 Algorithmic Load Distribution

The computational display design framework allows the use of parallel, distributed or single channel processing for image generation. We use the former for the Multi-Planar display. Each facet of the display in our design is independent, however, to retain the interactivity of the application we do not render to each facet independently. Parallelism is exploited at the facet and the scene triangle level. This provides with a framework that scales well to large display configurations and also maintains interactivity. Rendering however, occurs simultaneously - in which all facets are rendered in a single pass of rasterization. The design and implementation of these algorithms follow.

2.4 The Multi-Planar Display Framework

We describe the design, scaling and the rendering to multi-planar displays in this section. A general multi-planar display is piecewise planar, consisting of a number of polygonal facets forming the shape. For instance, one's personal workspace or desk could be a multi-planar display with horizontal, vertical, and slanted facets. In our design, each display facet is built using an LCD panel, micro-projector, or a suitable display mechanism and can show stereoscopic 3D based on the technology used.

A standard polygonal mesh model describes the display geometry in our scheme, with any number, shape, and size of planar facets. Each pixel of the display is addressed using a three-dimensional coordinate (f, i, j), where f is the facet id, ranging from 1 to the number of facets, and (i, j) is the pixel id within the facet. Facets can be controlled independently using a graphics channel for each. This, however, does not scale well to a large number of facets. Multiple facets can be driven together by assigning a mapping from each facet to a rectangular quilt image, generated by a single graphics channel. By segregating facet packing into multiple quilt images, quality can be improved further if needed. Un-mapping of facet images given the quilt image can be achieved using multiple VGA outputs or specialized hardware. We consider the un-mapping to be a part of the display hardware and is defined as part of the *display configuration* along with the facet geometry (Figure 2.2). This scheme ensures quality rendering on display facets as the facet resolution dictates the size and number of quilt images needed and not vice-versa, thus maintaining pixel density on each facet.

The display configuration, scene to be rendered and the user location are inputs to our system. The system generates the quilt images based on the user location and sends these to the display hardware for un-mapping. The process is described in Figure 2.2. As the first step, triangles of the scene are sorted in parallel to bins corresponding to each display facet (Section 2.6). This helps scale our rendering mecha-

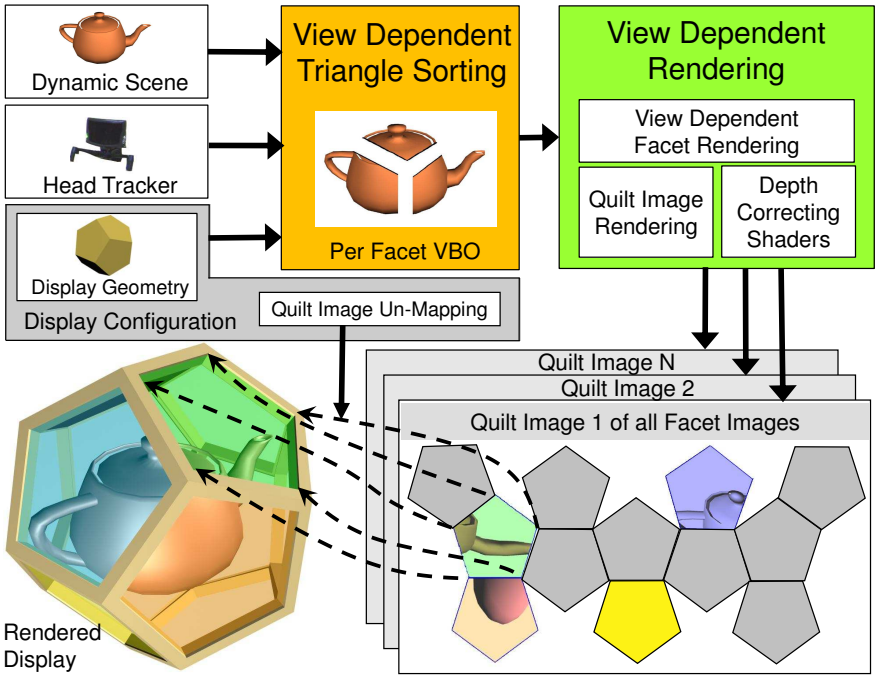


Figure 2.2 The multi-planar display system. Top: Rendering pipeline. Bottom: Driving the display using quilt images.

nism to arbitrary shapes, reducing the rendering load per facet and retaining interactivity of dynamically changing scenes. Next, the scene is rendered using facet-specific homography and per-pixel depth correction, which guarantees a consistent, artifact free view of the scene on the display (Section 2.5). A pre-warping based quilt image generation facilitates single-pass rendering (Section 2.6.2). The system is implemented using CUDA GPUs, which perform both the parallel sorting and rendering in a single pass of rasterization. The presented pipeline generates quality views from the user's perspective at interactive frame rates and can also be used for an inside-to-outside display configuration such as the CAVE.

2.5 Accurate View Dependent Rendering

In this section, we describe a novel scheme to render to multi-planar display facets. Our scheme generates quality images per facet without artifacts, eliminating inconsistencies across facet boundaries. Quality comparison with other rendering methods is also presented.

Given the viewer position and a look-direction, a symmetric frustum about the view direction preserves maximum detail. A virtual viewer camera C_v can be assigned to the viewer for generality. The position of C_v is tracked and known in a common frame of reference to that of the display. A *virtual*

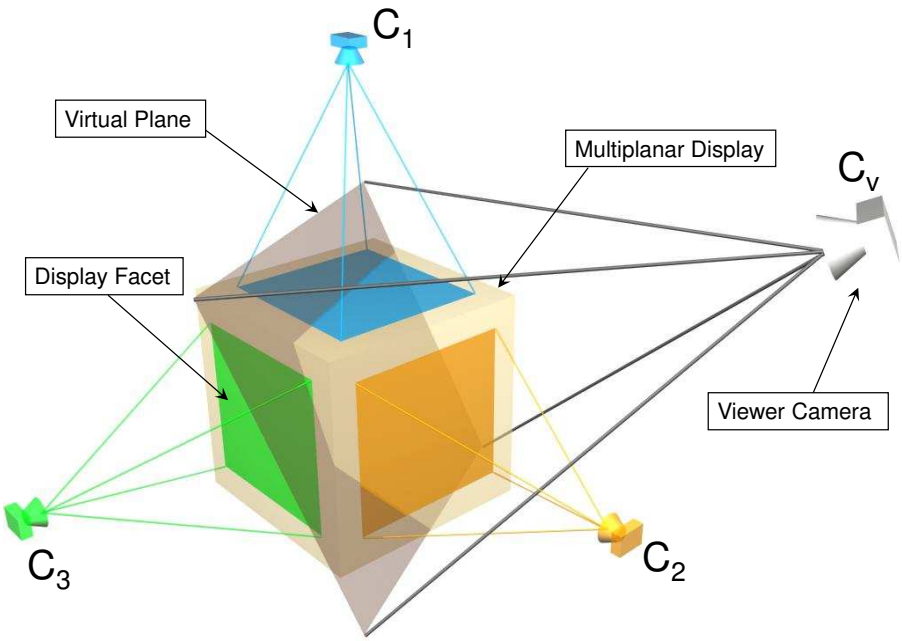


Figure 2.3 Viewer camera C_v viewing the display and the *virtual* plane passing through the center of the display. C_1, C_2, C_3 are cameras corresponding to each display facet.

view plane normal to the view direction passing through the center of the display can also be assumed without loss of generality, as shown in Figure 2.3.

The view-dependent virtual plane intersects multiple facets of the display. Each facet f can be assigned a facet camera C_f; its image I_f can then be related to the viewer camera image I_v by a 3×3 planar homography matrix as given below [52].

$$I_f = H_{fv}\, I_v, \quad \forall \;\; f \leq \#facets \tag{2.1}$$

Each visible facet will have such a homography relating its image to the virtual plane. Rendering I_f collectively using the above equation produces the illusion of the image lying on the virtual plane when observed from C_v. To ensure exact rendering, we render I_f such that the appearance and depth corresponds to C_v at every visible pixel.

Computing Facet Homography The display exists in physical world and its corner's 3D coordinates are known with respect to a fixed origin. Viewer's head position is also tracked in this coordinate system. Using the extrinsic and intrinsic camera parameters of the viewer camera C_v we can project the corners of the cube on the viewer camera's image plane, let these coordinates be (x_1, y_1), $(x_2, y_2)...(x_m, y_m)$. For each facet the image to be shown is stretched over the entire screen. Using image space coordinates we can relate the 2D points $(x_1, y_1)...(x_m, y_m)$ to the LCD screen's coordinates as shown in Figure 2.4.

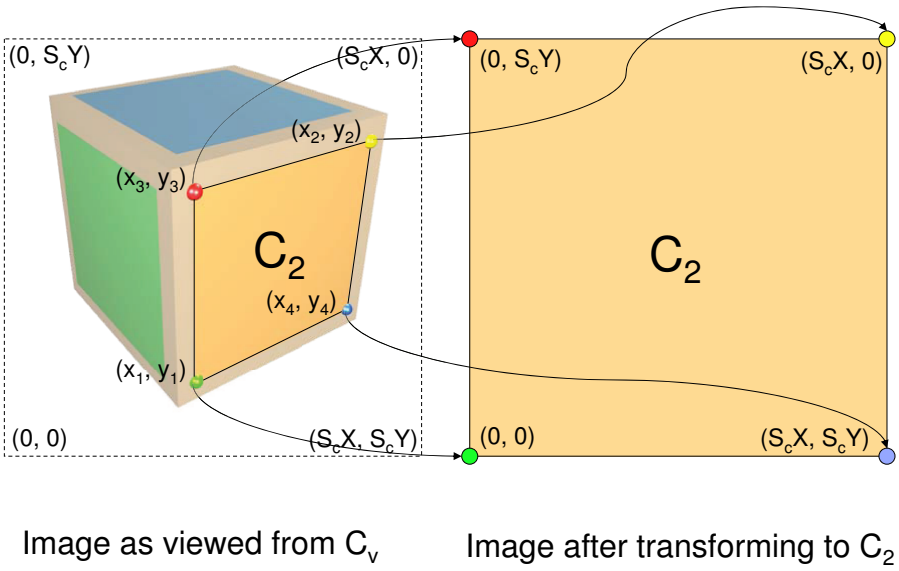


Figure 2.4 Point Correspondence for homography computation

These correspondences can then be used to compute the homography transformation required to convert image on the virtual plane to image on each facet of the display. This can be done by solving the following set of equations using the algorithm given in [52].

$$\begin{bmatrix} a \times 0 & b \times S_cX & c \times 0 & d \times S_cX \\ a \times 0 & b \times 0 & c \times S_cY & d \times S_cY \\ a & b & c & d \end{bmatrix} = H_{fv} \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

Where a,b,c and d are the unknown scale factors within the homography matrix.

Alternatively, given the rotation matrix R and the translation vector T between C_f and C_v, normal n, distance d of the virtual plane, and intrinsic parameters K_f and K_v of the cameras, H_{fv} can be computed directly as $H_{fv} = K_v[R - Tn^T/d]{K_f}^{-1}$ [52] (Figure 2.5). We compute the homography for each facet independently in each frame based on the viewer position. Homographies are computed in parallel on the GPU from known information about C_v and C_f using a thread for each facet.

2.5.1 Rendering Using Image Space Homographies

OpenGL follows a four stage transformation pipeline (Figure 2.6). A 3D point is transformed from world coordinates to camera coordinates using the model and view transformations, M_v. This is followed by converting the camera coordinates to image coordinates using the projection transformation,

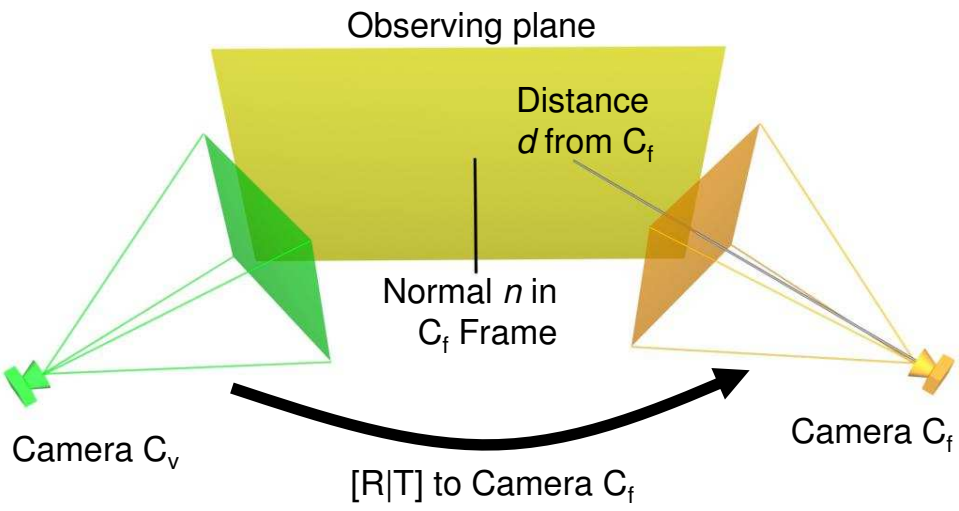


Figure 2.5 Computing Homography transformation between two cameras.

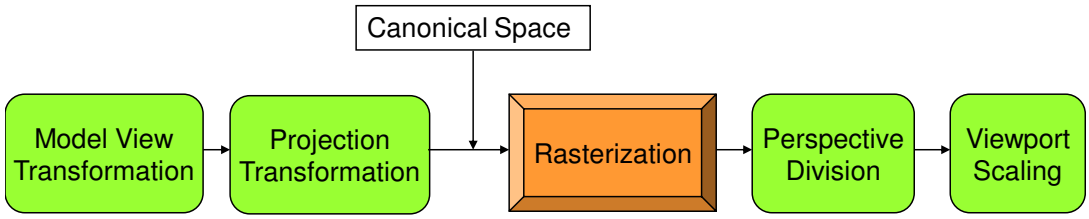


Figure 2.6 Graphics pipeline with canonical space. The perspective division and viewport scaling stages only modify the scale of the point in canonical space.

P. Thereafter the 2D points are normalized using perspective division, P_d, and then scaled to the desired view-port, V. For the virtual plane rendering of a 3D point $\vec{X}$ can be written as follows.

$$I_v = VP_dPM_v \vec{X} \tag{2.3}$$

As homographies are computed using image coordinates we must apply them to images only. That is, for converting image I_v to I_f using H_{fv}, image I_v must first be rendered. Thus even after combining equations (2.1) and (2.3) we do not gain much in terms of single pass rendering when using Equation 2.4.

$$I_f = H_{fv}VP_dPM_v \vec{X} \tag{2.4}$$

2.5.2 Problems Using Image Space Homographies

Though this method works in theory there are many practical pot holes. Image I_v must first be rendered before homography H_{fv} can applied to it. H_{fv} must be applied to every pixel of I_v to get the desired image I_f. Since the computed homographies are only up to scale, i.e. the last row of homographies in equation (2.4) is not of the form [0 0 1], there exists a per pixel scale value as a result of multiplying the homography to image coordinates, as shown in equations (2.5) and (2.6)

$$\begin{bmatrix} cx' \\ cy' \\ c \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.5}$$

where c is the per pixel scale factor depending on the pixel coordinate

$$c = h_{31}x + h_{32}y + 1 \tag{2.6}$$

To generate the desired image normalization at every pixel is needed, which can be done using a pixel shader. Image coordinates are transformed using this method. After applying the homography and normalization, we may not end up with integer pixel coordinates, quantizing which can result in artifacts in the transformed image. Further, as only rendered pixels are transformed, after transformation there may exist holes in the transformed image which cannot be filled using the original image as there may not exist a corresponding pixel in the original image. Camera-projector systems deal with this problem by interpolating pixels to fill the holes and quantizing coordinates to integer values, which results in blurring and other artifacts in the final image. However, since our image is a rendered image, we can eliminate the need of such pixel operations by integrating homography into the rendering process itself.

2.5.3 Computing and Using Homographies in the Canonical Space

Coordinates in the Canonical space lie in the interval $[-1,1]$. This normalized space lies just before the perspective division stage in the OpenGL pipeline (Figure 2.6). A 3D point when passed through the model-view and the projection transformations results in a canonical space point.

Computing homographies in this space is equivalent to image space homographies, as the relative position of pixels do not change later in the pipeline. Computing homographies in this space still results in per pixel scale value, however, as this space lies before the perspective division stage the effect of the scale factor is nullified. Furthermore, exclusive quantization of coordinates to integer values is not required as that is automatically done during the view-port scaling stage of the pipeline.

Both problems of image space homographies are eliminated if homographies are computed in the canonical space. This idea was outlined in the context of correcting for off-axis projection by Raskar [88]. To achieve this the intrinsic parameters of viewer's camera C_v must change such that the projection of dispaly's corner coordinates are in the range $[-1,1]$ as oppose to $[0 - S_cX, 0 - S_cY]$. Let these coordi-

nates be (x'_1, y'_1), $(x'_2, y'_2)...(x'_m, y'_m)$. The corresponding screen coordinates of each panel should also lie in range [−1,1] as oppose to $[0 - S_cX, 0 - S_cY]$.

Using these coordinates and their correspondences one can solve the following set of equations for homography transformation for each facet of the display.

$$\begin{bmatrix} -a & b & -c & d \\ -a & -b & c & d \\ a & b & c & d \end{bmatrix} = H_{fv} \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 \\ y'_1 & y'_2 & y'_3 & y'_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.7}$$

To render the image, we must multiply the homography matrix prior to perspective division stage i.e. after the projection matrix as the homography is computed in this space only. Thus final rendering can be done using the following equation for each facet of the display.

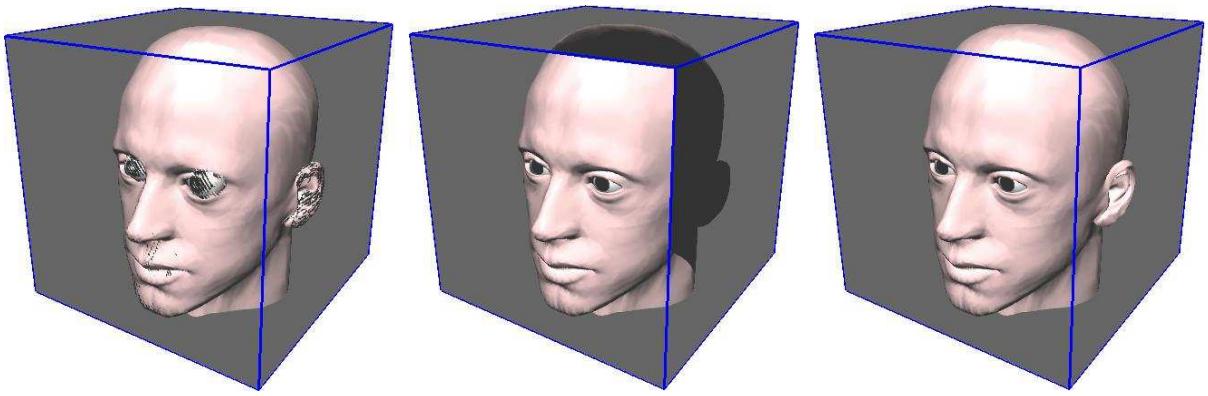
$$I_f = VP_dH_{fv}PM_v \vec{X} \tag{2.8}$$

Depth Correction Though the above mentioned rendering scheme renders correct image for each facet of the display, after using homography the depth values per pixel may not lie in the range [−1,1] that can lead to pixel dropping and clipping of geometry by the near and far planes. Depth values at pixels may go out of the canonical space range of $[-1, 1]$ using the above scheme. For a point $(X_c, Y_c, Z_c) \equiv M_v\vec{X}$ in the camera frame when multiplied by projection and homography matrices, the effect can be understood as follows:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & 0 & h_{13} \\ h_{21} & h_{22} & 0 & h_{23} \\ 0 & 0 & 1 & 0 \\ h_{31} & h_{32} & 0 & 1 \end{bmatrix} \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix} = H_{fv} \begin{bmatrix} A & 0 & B & 0 \\ 0 & C & D & 0 \\ 0 & 0 & E & F \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = H_{fv}PM_v\vec{X} \tag{2.9}$$

The matrix shown to the left of $[X_c\ Y_c\ Z_c\ 1]^T$ is general projection matrix used in the OpenGL pipeline. We observe the final depth value z'/w' after applying the homography may not lie in the $[-1, 1]$ range as $I_z \equiv z'$ (depth without homography) belongs to $[-1, 1]$. After applying homography w' becomes a function of I_x and I_y, then z'/w' need not lie in the range [−1,1] leading to intersections at $z =$ near and $z =$ far for some pixels. Uniformly scaling the depth by a z-scale factor less than 1 can bring the depths within range, but will not guarantee correct ordering. This can lead to poor depth resolution and push through artifacts as seen in Figure 2.7(a). Raskar [88] suggests a scale factor of $(1-|h_{31}|-|h_{32}|)$. This reduces the depth resolution and can suffer from near plane clipping. Multi-Planar displays can have serious artifacts at facet junctions using this method as shown in Figure 2.7(b).

Both the above mentioned solutions to the depth problem approximate the depth buffer of the virtual plane. Approximations do not suffice for applications requiring accurate rendering. Furthermore, use of these approaches can lead to inconsistencies across display facets that can seriously compromise the overall visual quality of a multi-planar display. We solve this problem exactly by scaling each pixel independently using a pixel shader. We see depth at a pixel, z_p, can be written as:



(a) Scale < 1 (b) Raskar (c) Our method

Figure 2.7 The depth problem: comparison of approaches. (a) depth errors due to reduced z-range at eye, lip and ear. (b) error due to near plane clipping artifacts. (c) our method without artifacts

$$z_p = \frac{z'}{w'} = \frac{I_z}{w'} \notin [-1, 1] \tag{2.10}$$

However, without using the homography, the correct depth buffer value is

$$z_v = \frac{I_z}{I_w} = \frac{I_z}{-Z_c} \in [-1, 1] \tag{2.11}$$

In order to convert z_p to z_v we multiply each pixel's depth value z_p by w'/I_w after applying the homography matrix. The correct depth buffer value can be computed as follows

$$z_v = \frac{z_p w'}{I_w} = \frac{z'}{I_w} = \frac{z'}{-Z_c} \tag{2.12}$$

The depth buffer for each facet will have the same depths from the viewer camera C_v. The depth resolution is exactly same on all facets, ensuring uniformity at facet junctions. We change the depth of each pixel in a fragment shader, which has access to z'. The Z values of the vertices are sent by the vertex shader and are interpolated to make Z_c available at each pixel. The shader computes z_v as a ratio of these two and sends it to the framebuffer. Figure 2.7(c) shows the correct image using our scheme. Early-z culling is avoided by setting a constant depth value of -1 at all vertices. Rasterization is not affected by this and the correct depth values are computed later by the fragment shader. Algorithm 1 outlines the vertex and fragment shaders used in our rendering scheme.

2.5.4 Comparison with Other Approaches

We compare our rendering scheme with alternative methods used to render to FTVR displays. The goal of our system is to provide quality rendering at interactive frame rates. The comparison thus focuses on quality, primarily on the facet. Off-axis-rendering renders the scene directly to the facet and thus is capable of producing quality images. It is the primary alternative to our method. Projective Texture

Algorithm 1 Depth Correcting Shaders

1: {**Vertex_Shader(V)**}
2: Perform fixed-pipeline operations
3: Compute camera space vertex coordinates V_c
4: Send V_c to pixel shader with interpolation
5: Set z coordinate of output vertex as -1
6: {**Fragment_Shader(V)**}
7: Perform fixed-pipeline operations for color
8: Transform V_c to canonical space as V_c'
9: Set depth as the ratio of z coordinate of V_c' and V_c.

Mapping (PTM) can also be used to generate a similar effect. It produces an intermediate image which is mapped to multiple facets using interpolation. This can lead to quality degradation due to image re-sampling. A detailed quality comparison of our method with Projective Texture Mapping can be found in the Section 2.12 at the end of this chapter.

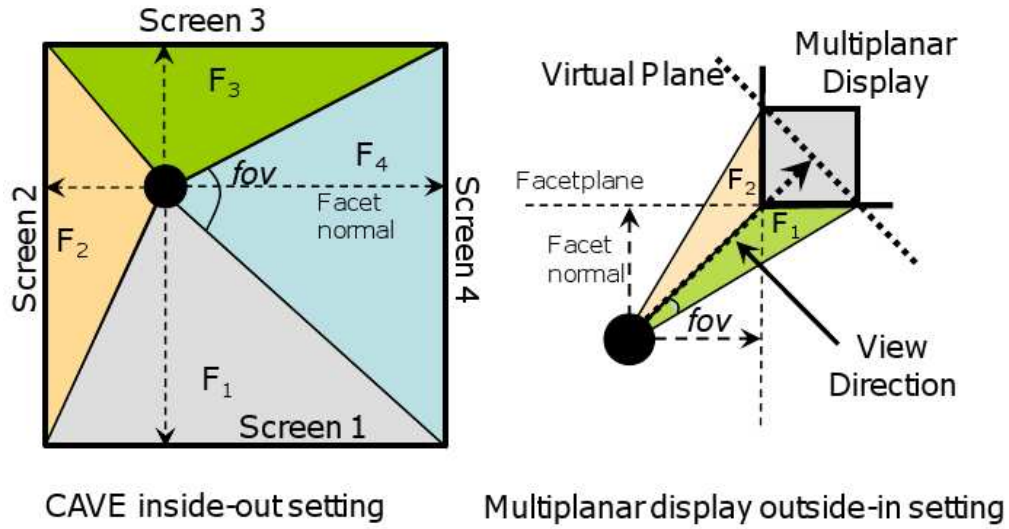
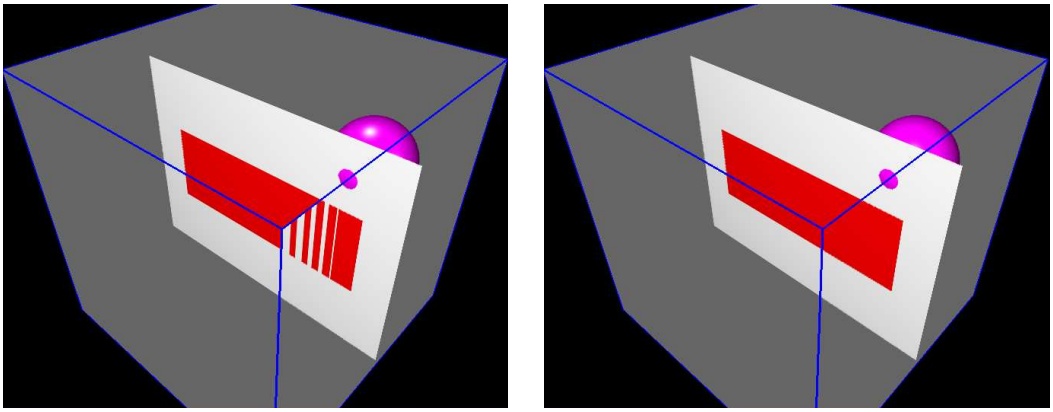
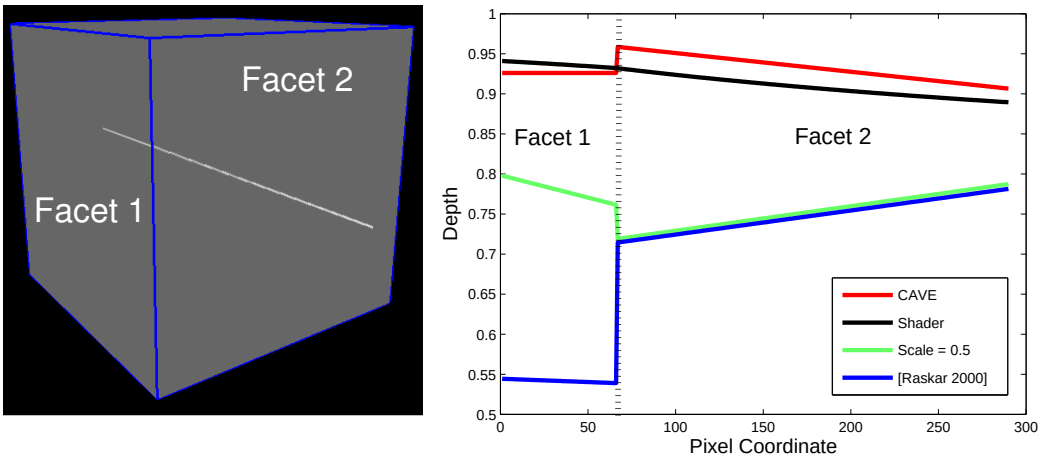


Figure 2.8 Top view of CAVE and multi-planar display setups with off-axis frustas. The view angle is large for multi-planar display using this approach.



(a) Off-Axis (b) Our method

Figure 2.9 Comparing off-axis projection with our rendering scheme. Scene consists of two proximate rectangles and a push-through sphere.



(a) The scene (b) Depth Buffer

Figure 2.10 Comparing depth buffers for various approaches at the boundary of two facets. Note no sudden change in depth values for the shader approach, ensuring continuity at facet boundaries.

Off-axis rendering [38] has been used as the rendering mechanism for FTVR displays like the CAVE, Cubby, etc. The method renders each facet using a view plane parallel to the facet using oblique frustum boundaries (Figure 2.8). The rendering is not geometrically correct due to depth errors. Depth for each facet varies with the viewer location and is not consistent across facet boundaries. Akeley and Su report that the error increases linearly with the angle of view and is particularly large if the scene resides in corners of the frustum [14]. In a display configuration like the CAVE, this angle of view is small and errors are rarely visible. In an outside-to-inside configuration, the scene always resides in one corner of the frustum and the angles could be large (Figure 2.8). This results in visual artifacts that can be seen across facet boundaries. Figure 2.9(a) shows the rendering of two proximate planes using off-axis

rendering. Depth errors can be seen as opposed to our rendering scheme (Figure 2.9(b)). Figure 2.10 shows depth variation for various methods across a facet boundary for a scene consisting of a single line. We see that only our method ensures consistent views at the junction of two facets. Rendering to Multi-Planar displays requires this property to avoid artifacts at facet boundaries.

2.6 Scaling to a Large Number of Display Facets

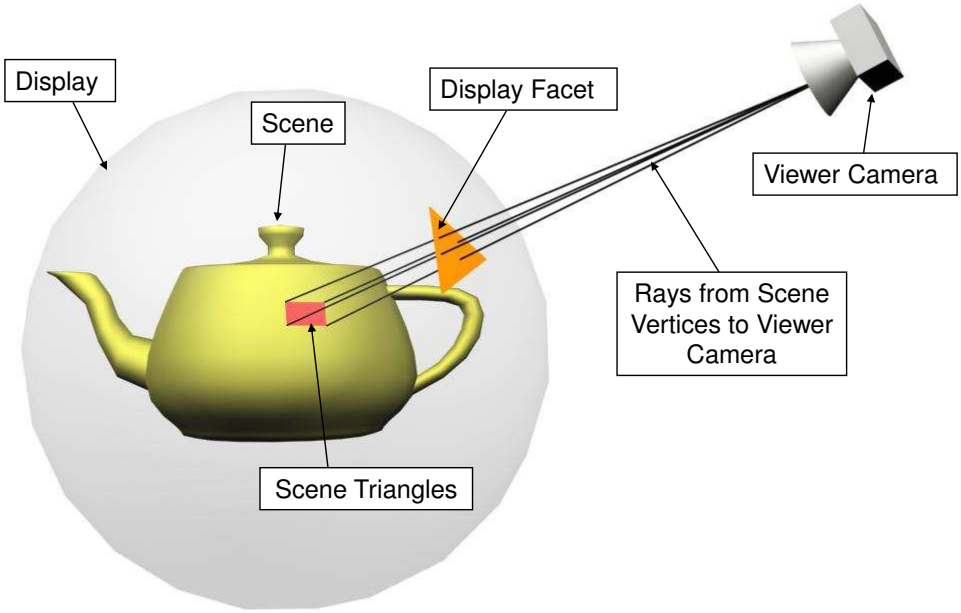


Figure 2.11 Triangle sorting based on per vertex raycasting

Current FTVR rendering methods render the scene to each facet independently. This leads to a linear increase in the rendering load with the number of facets and may become impractical beyond a few tens of display facets. We propose two load balancing ideas to aid our rendering scheme in this section: (i) A parallel scene sorting algorithm which sorts the triangles of the scene to the display facets quickly on the GPU per frame. (ii) A rendering scheme that renders the quilt of multiple facet images in a single pass of rasterization. These achieve real-time rendering of fully dynamic scenes for displays consisting of thousands or more facets. Methods based on spatial hierarchies have been shown to cull the scene to reduce the rendering load per facet but are hard to extend to dynamic/deforming scenes due to their static design. Because of our parallel design, our method outperforms spatial hierarchies as reported in Section 2.9.

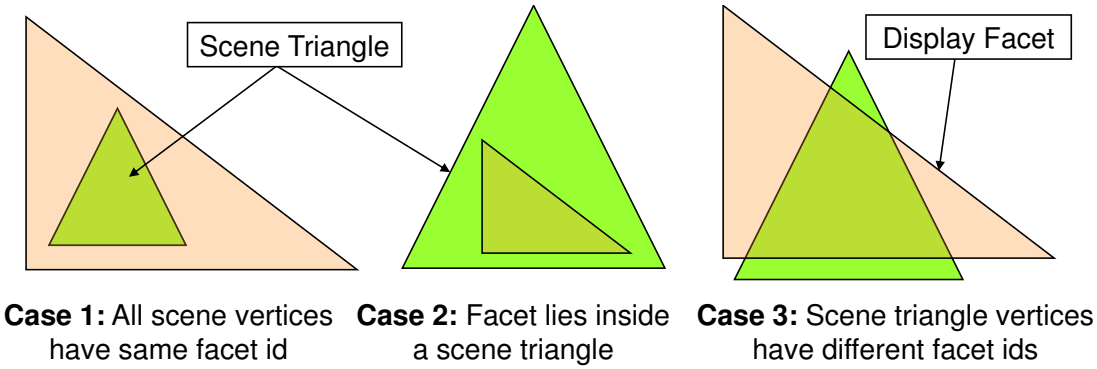


Figure 2.12 Cases occurring from per vertex ray casting

2.6.1 Visibility Determination and Triangle Sorting

We sort the triangles of the scene in parallel to respective facets of the display per frame. This provides flexibility to our system in handling dynamically changing scenes. The parallel implementation also scales well to a large number of display facets. Per vertex ray-casting implemented on the GPU is used to achieve this in real-time. Figure 2.11 shows our sorting approach for a large number of facets. A ray is shot from the viewer location to each scene vertex. The intersection of the ray with a given facet of the display determines its facet id. The information is stored on the GPU memory in a temporary array *facetid*. Three situations can arise from per vertex ray casting as shown in Figure 2.12. Case 1 is the most frequent case. The second and third cases occur for fewer number of triangles, but require more expensive computations. We divide our sorting into two passes based on this observation. The first pass handles the first case while the second pass deals with the remaining triangles.

Pre-Processing and Optimizations An octree structure is used to reduce the number of tests that needs to be computed per ray. Since the display geometry is static we pre-process the facets into an octree spatial data structure, with each octree leaf assigned display facets it intersects spatially. We use a depth level 3 with 512 leaf nodes, stored as an array in the GPU shared memory for fast access [85]. Each ray first emulates the octree leaf nodes it passes through and performs the relative tests with those facets only. This reduces the number of tests performed per ray potentially by $1/8^n$, where n is the octree depth.

We also reduce the work done by the GPU for second pass of the algorithm by reducing the number of threads executing in parallel for the second pass, Figure 2.13. In a temporary array, of size of number of scene triangles, a 1 is placed where first pass has failed, and 0 otherwise. We apply a parallel scan [106] operation on this array which yields the position of the new threadID that needs to execute the second pass. A simple kernel then compacts these entries into a new array and keeps a mapping for threadIDs

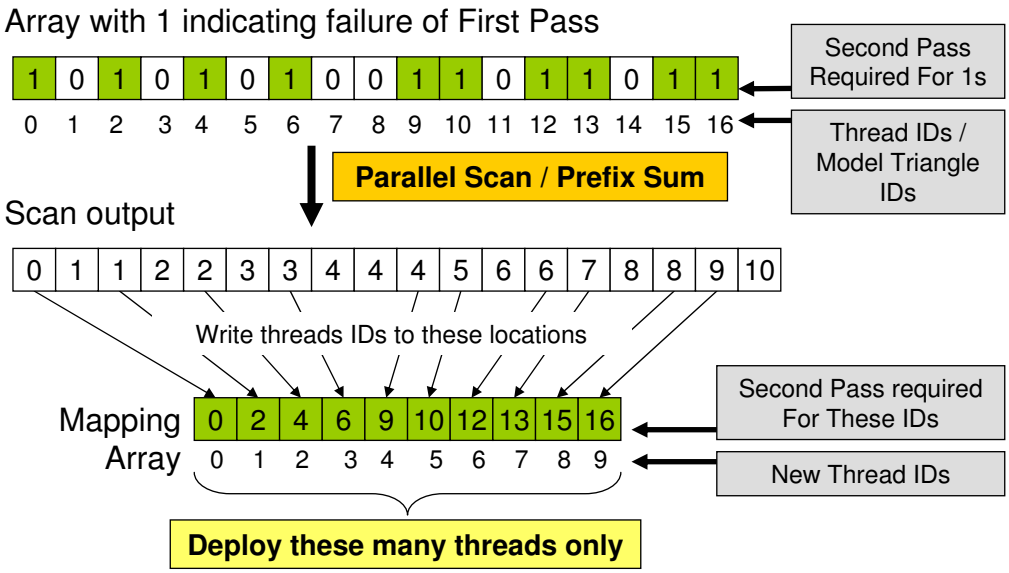


Figure 2.13 Reducing the number of threads for the second pass of the culling algorithm

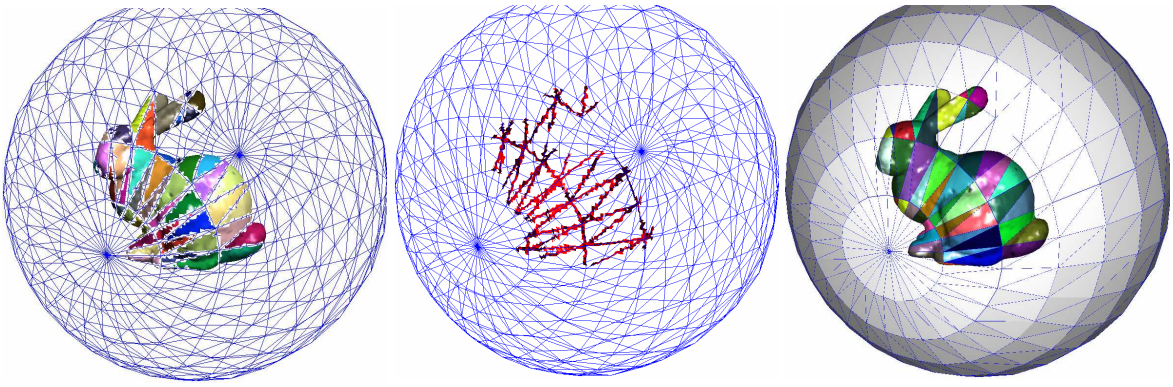
to scene triangles IDs. We execute the second pass with only as many number of threads as present in the new mapping array.

First Pass The first pass assigns a facet id to triangles based on facet ids of its vertices. Each triangle gets assigned a single facet id if all its vertices have the same facet id, as shown in Algorithm 2. The scene triangle id and its facet id are stored as a tuple in a temporary array TFID on the GPU, which is later used to bring all triangles of a facet together. For typical scenes, this pass assigns facet ids to 80% to 90% of the total scene triangles. The remaining triangles have different facet ids at their vertices and undergo the second pass. Figure 2.14(a) depicts triangles passing the first pass for the Bunny model on a spherical display.

Algorithm 2 First Pass

1: $tid \leftarrow GetThreadID()$ {**thread id = triangle id**}
2: $count \leftarrow$size_of(TFID)
3: **if** for all vertices vid of triangle tid, $facetid[vid]$ is same **then**
4: TFID[$count$]$\leftarrow$(tid, $facetid[vid]$)
5: $count \leftarrow count + 1$
6: **end if**

Second Pass The second pass examines all triangles that remain (Figure 2.14(b)). In this pass, scene triangles and display facets are projected using viewer camera, and a 2D triangle-facet intersection test is



(a) First Pass (b) Second Pass (c) Rendering

Figure 2.14 The stages of our triangle sorting algorithm

performed using Möller's procedure [77]. Overlap test shown in case 2 of Figure 2.12 is also performed. Facet ids are assigned to the triangle if any of the tests are positive. Multiple facet ids may be assigned to a single triangle during this pass. Each such triangle-facet ids tuple is added to the TFID array. Algorithm 3 summarizes these steps.

Algorithm 3 Second Pass

1: $tid \leftarrow GetThreadID()$ {**thread id = triangle id**}
2: $count \leftarrow$size_of(TFID)
3: Project scene triangle using Viewer Camera C_v
4: **for** all facets $fid \in$ Emulated Octree Nodes **do**
5: Project facet fid using viewer camera C_v
6: **if** triangle-facet intersection [77] OR projected facet lies inside projected scene triangle **then**
7: TFID[$count$]$\leftarrow$(tid, fid)
8: $count \leftarrow count + 1$
9: **end if**
10: **end for**

The TFID array holds the triangle–facet id tuples for all possible triangle-facet intersection cases after the second pass. We bring all triangles belonging to a facet together by performing a split [86] operation on TFID using the facet id as the key. A kernel runs over the length of the split TFID array and copies the scene triangle data to the independent VBOs per facet in parallel. VBOs thus created undergo the rendering process reported in Section 2.5 to produce accurate views for each facet as shown in Figure 2.14(c), with colors indicating different facet ids.

The Scene Sorting Procedure The overall sorting procedure is described in Algorithm 4. Our algorithm increases the number of triangles to be rendered, replicating triangle data to facets in which they appear during the second pass, but this increase is small in practice and empirically the rendering is

faster than both spatial hierarchies and independent rendering for each facet as reported in Section 2.9. An average increase of $4-5\%$ is seen for the number of triangles.

Algorithm 4 The Scene Sorting Algorithm

```
1: for all scene triangles in parallel do
2:     Perform the first pass
3:     Compact triangle ids that failed the first pass
4:     Perform the second pass
5:     Store scene triangle-display facet pair in TFID
6: end for
7: Split TFID using display facet as the key
8: for all entries of TFID in parallel do
9:     Copy scene triangle data to display facet id VBO
10: end for
```

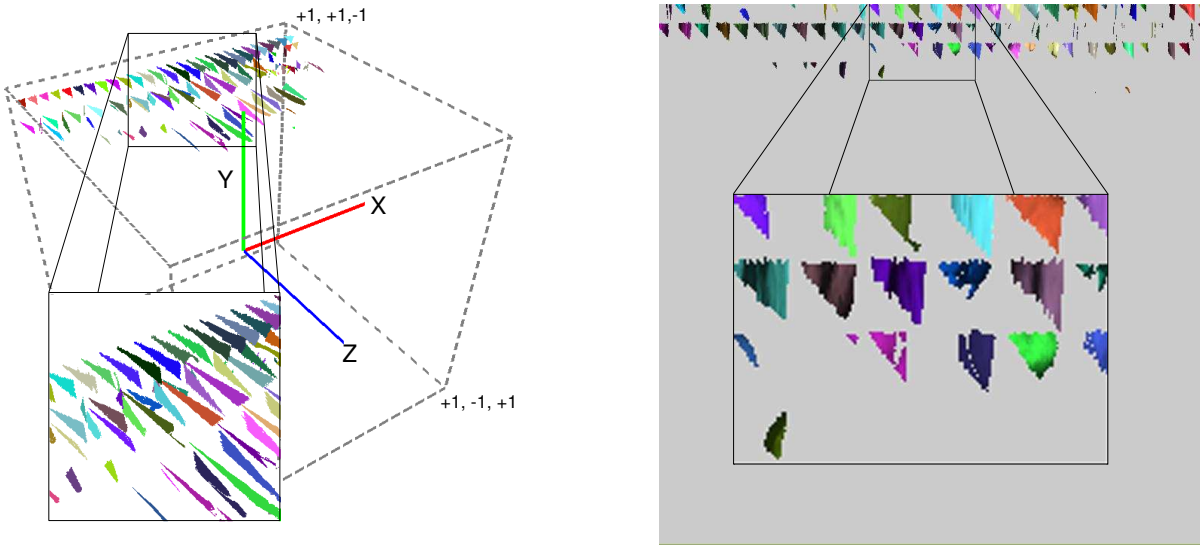
2.6.2 Rendering the Quilt Image for all Display Facets

In this section we describe a method for reducing the facet image rendering load for a large number of facets. Rendering to each facet independently requires a setup phase of the graphics pipeline. This overhead can increase significantly as the number of facets increase. For speed, we render all facet images in a single pass of rasterization. A trade-off exists between the required resolution and number of rasterization passes, as high facet resolution can dictate facet packing into multiple quilt images – each requiring a separate rasterization pass. An optimal packing is thus needed to utilize the quilt image space efficiently.

From Equation 2.8, we observe that each display facet has a different projection matrix ($H_{fv}P$). The difference in projection matrix warrants independent rasterization, setting independent viewports and projection matrices for each facet. This can slow the rendering as the number of facets increase. We can avoid this by pre-warping each point $\vec{X}$ for each facet. The mapping for a facet image f in the quilt image is a fixed 2D transformation (Figure 2.2). Let Y_f denote this in the canonical space. The point $\vec{X}$ per facet is modified by the facet-specific quilt image mapping (Y_f), homography (H_{fv}) and the projection and modelview matrices (PM_v) to $\{Y_f H_{fv} P M_v \vec{X}\}$. This modified point can be thought of as a point in another canonical space with its modelview and projection matrices set as identity. The point in this space depends on facet id f, but the pipeline parameters are same for all facets. This enables the same rendering pipeline to render all facets to the quilt image in a single pass of rasterization using

$$Quilt = V P_d I_P I_{M_v} \sum_{f=1}^{\#facets} Y_f H_{fv} P M_v \{VBO_f\}, \tag{2.13}$$

where V is the viewport for the quilt image, P_d the perspective division stage and I_{M_v} and I_P the modelview and projection matrices respectively set to identity.



(a) Scene in Canonical Space

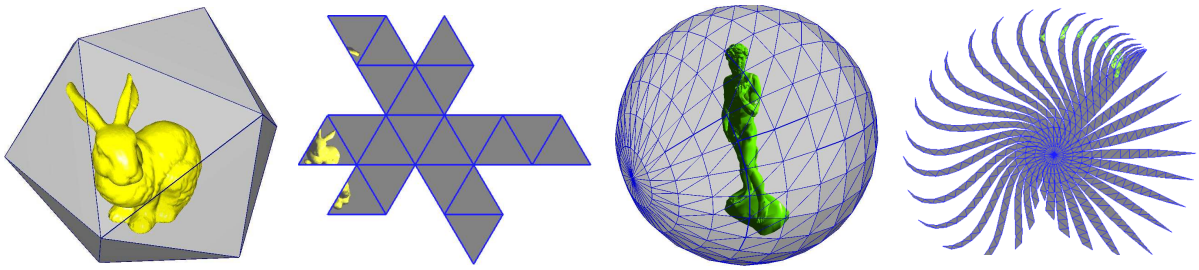
(b) Rendered Quilt Image

Figure 2.15 The canonical space triangle separation and the corresponding rendered quilt image

A vertex $\vec{X}$ in the list of facet f can be pre-warped to $Y_f H_{fv} P M_v \vec{X}$ after the scene sorting stage. The facet VBOs created at the end of the sorting phase are thus modified while copying the data from the TIFD array. Figure 2.15(a) shows such a transformed scene in the canonical space, with colors indicating facet ids. It can be seen that VBOs deform under these transformations. However, projecting them to the quilt image generates the correct image per facet as shown in Figure 2.15(b). For a display with 1600 facets, a $15 - 20\%$ decrease in rendering time is seen using this method compared to setting up the graphics pipeline for each VBO at the same facet resolution.

2.6.3 Mapping and Un-Mapping of Facets In the Quilt Image

Facet mapping transformation Y_f is a 2D transformation matrix per facet that maps the facet pixels (f, i, j) to the pixels in the quilt image (u, v) as shown in Figure 2.2. This can be done without any interpolation using equal number of pixels in each. Y_f differs from texture mapping as a result. Optimal mapping of polygonal shapes into a 2D plane is an NP hard problem [69]. We can use a heuristic mapping arrangement that attempts to utilize the quilt image space optimally. This is a preprocessing step done while designing the display configuration. A simple mapping algorithm would be to assign a bounding rectangle to each facet polygon and pack them in a linear order across the dimensions of the quilt image, as used in Figure 2.15. This scheme may not utilize the image space efficiently. Better packing using a BFS tree can be used to generate a tighter quilt image by expanding the facets of the display according to a BFS tree which can then be embedded on the quilt image plane, as shown in Figure 2.16.



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

Figure 2.16 Figures (a) and (c) show the display as seen from user's point of view with their respective BFS expanded quilt image given in Figures (b) and (d).

Un-mapping the quilt image to display facets is inverting Y_f, i.e. mapping (u, v) pixel in the quilt image to a facet pixel (f, i, j). Given Y_f this can be achieved either using VGA outputs or specialized hardware. For example, for our LCD prototype (Figure 2.21), we render a large image comprising of 4 rectangles with same resolution as each LCD facet, placed in each corner. The display driver is configured to map these regions to the LCD displays using VGA outputs as used in multi-headed displays. Our system, however, treats the entire region as the quilt image. For generic shapes, the mapping is more involved. Electronics needed to send pixels to facets using the display configuration is easy to build. This can best be thought of as part of the display hardware, such that a rectangular quilt image can drive any multi-planar display.

2.7 The Rendering Pipeline

Figure 2.17 describes the overall rendering process for our multi-planar display system. The scene being displayed can be completely dynamic since triangle sorting and VBO creation takes place every frame. We can use one or more commodity GPUs for the processing and rendering, keeping the cost low. Our setup allows us to render scenes with up to 200K triangles in real-time to a display made up of up to 1600 facets on the Nvidia GTX 580 GPU. For larger scenes the framerate drops to below interactive rates as shown in Figure 2.28. Multi-GPU solutions can be used to enhance the speed in such cases, as the problem is parallel-friendly. Distribution of the quilt image to physical facets can be done using appropriate electronics in the display system [112] or using a system with multiple display channels and coordinated rendering. We use VGA outputs and texture mapping for our prototype displays.

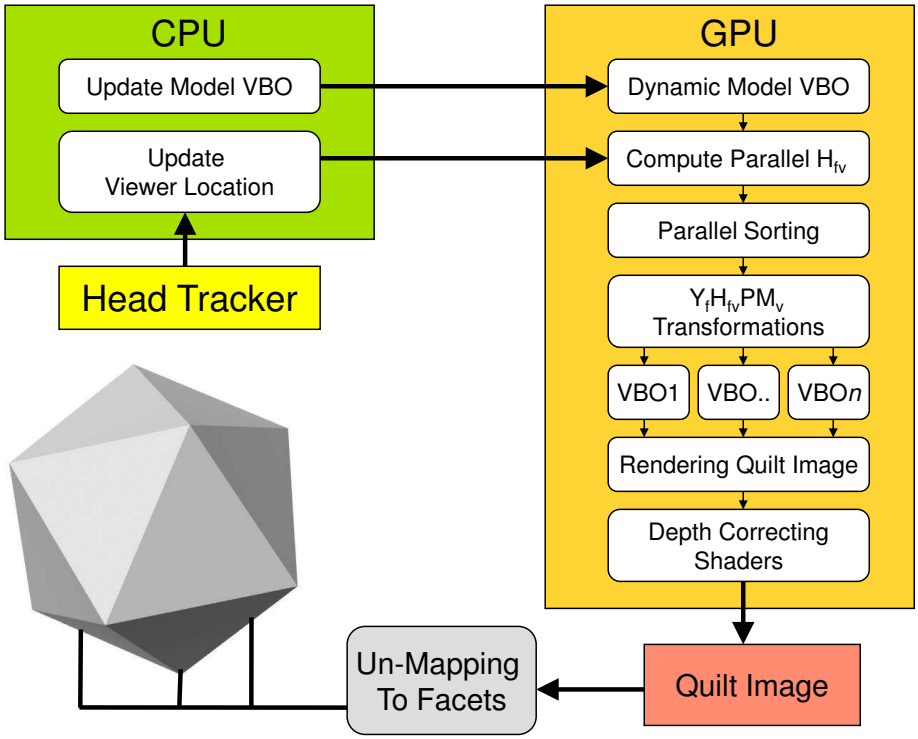


Figure 2.17 The overall rendering process of our multi-planar system.

2.8 Displays Prototypes

In this section, we describe prototype displays that we built and simulated to validate our multi-planar display framework. Building a multi-planar display of a general shape requires engineering at the display device level that only a display company can undertake. The objective of considering different prototypes is to establish (a) the generality of the multi-planar display framework, (b) the correctness of displaying 3D information using our method on challenging display configurations, and (c) the scalability of the approach to arbitrarily large displays. We use three types of display prototypes for this. Examples showing dynamic three dimensional scenes and information on our prototypes can be found in the accompanying video.

Tracking System The head location of the viewer is needed every frame to render to display facets. Many solutions are available for this purpose. A low latency head tracking system is required for good visual experience on fish tank based virtual reality displays. We use a infrared based head tracker, TrackIR5 [115], intended for use in gaming applications. With a refresh rate of 120Hz the latency is minimized and the location of the viewer is given within an error of $5-10\%$. Eye position is estimated

based on the viewer's head position, 10cm to the right and 5cm forward from the viewer's left ear position.

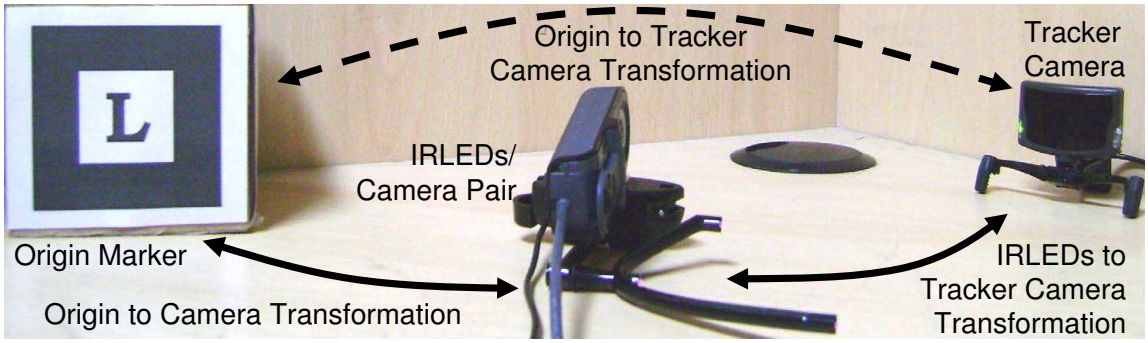


Figure 2.18 Calibrating the tracker with respect to a global origin

The tracker camera tracks three IR-LEDs worn by the viewer, and estimates the viewer's position based on it. We need to know the head position with respect to a global origin. We first calibrate the tracker camera. We setup the tracker camera in any orientation and estimate its configuration with respect to a global origin, marked by a visual ARToolkit [12] marker. We attach the IR-LEDs to a camera and move the setup simultaneously viewing the ARToolkit origin marker and the tracker camera (Figure 2.18). This provides the transformations of the IR-LEDs/camera pair with respect to the ARToolkit marker and the tracker camera. Transformation between the origin marker and the tracker camera can be estimated based on these matrices. We average over 100 such matrices to minimize the calibration error.

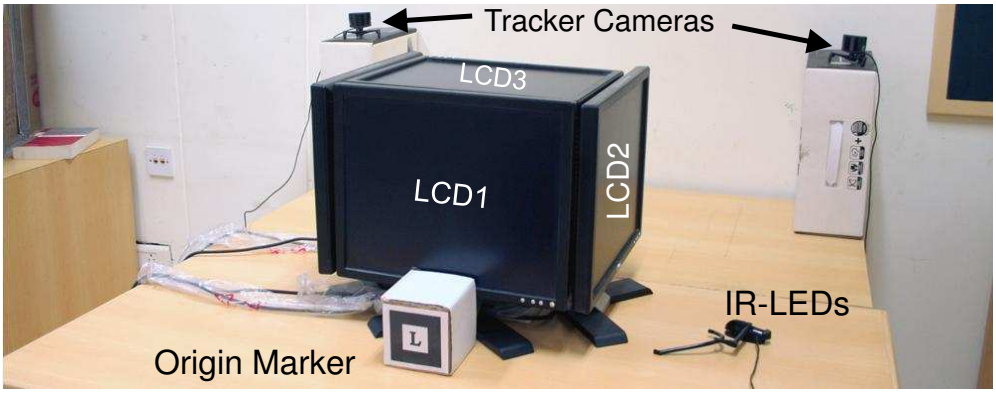


Figure 2.19 LCD based display configuration

2.8.1 LCD based Setup

We prototype a cube display using off the shelf LCD panels with up to 5 display facets located around the cube. The prototype is similar to the Cubee [113], but uses our rendering scheme as opposed to off-axis rendering. The display follows our rendering process with facet mapping achieved using VGA outputs. The cube is setup by loosely placing LCD panels in the desired configuration. Off the shelf LCDs are used to construct the display shown in Figure 2.19. We can take a fixed geometry file to specify the display or infer it using a calibration step.

Calibration We calibrate the cube using a simple procedure. The dimensions of the display area of the LCD panels are known. One printed ARToolkit marker is designated as the origin. Each LCD displays an ARToolkit marker at it very center, Figure 2.20(a). We establish the transformations from origin marker the facet markers by moving the camera slowly around the setup so that origin marker and facet markers are visible in several frames (Figure 2.20(b)). This helps recover the plane of the display and its center point. Combined with the display dimensions, the facets corners are now fully known. The cube can be calibrated in less than a minute, with no special hardware or equipment, Figure 2.20(c). We also provide tools to interactively adjust the calibration parameters, if the automatic process is not sufficient. Planes can be adjusted interactively to correct their positions.

(a) Facet Markers (b) Computing Transformations (c) Calibrated Display

Figure 2.20 Calibrating an LCD based display configuration

The procedure can be extended to a general polygonal display. If each facet is considered independent, as when built from independent LCDs, markers are needed on each facet. For rigid shapes markers need not be displayed on each facet. One of the facet markers will suffice for a set of facets if they follow a known rigid configuration. We use this procedure to calibrate projection based setups. Our calibration procedure is simple. Better methods and equipment can be used to calibrate the setup more accurately. The method, however, proves satisfactory and produces accurate enough calibration parameters for our prototype setups.

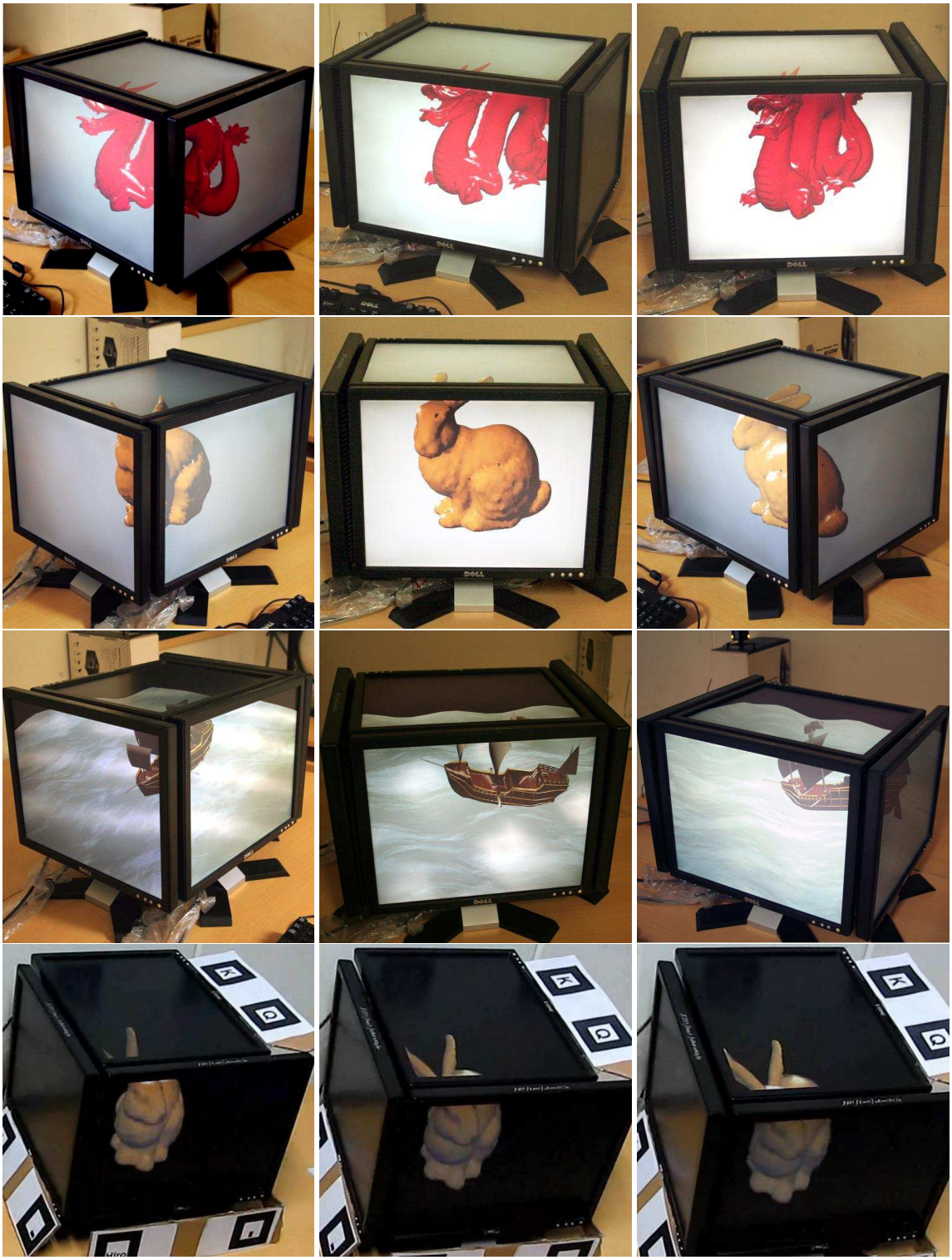


Figure 2.21 LCD based display showing various static and dynamic scenes

Brightness Correction Brightness/color correction is an important issue for tiled display setups [73]. For LCDs, intensity and colors fadeout with increasing viewing angle. We use a simple method to compensate for this; we change intensity of pixels based on the dot product of the facet normal and the view vector with maximum intensity at oblique viewing angles. The scheme produces a satisfactory visual experience.

Hardware Details of the 3D Cube We used a PC with two Nvidia Quadro FX 5600 cards to drive a 4-panel cube display. The display supports anaglyphic stereo display and monoscopic walk-around display. Shutter-based stereo using Nvidia 3DVision glasses can be built using high-frequency LCDs as the GPUs are genlocked. The LCDs have visible and thick borders, which affect the quality of view. However, the display areas are modeled correctly. Thus, the borders appear like supporting bars of the box in which the object is kept (Figure 2.21).

2.8.2 Projection based Setups

Since LCD based curved surfaces consisting of thousands of facets are hard to physically implement, we show the scalability of our approach using projection based setups. These setups use our pipeline to generate the quilt image by following the stages of parallel sorting, facet rendering, quilt image rendering and depth correcting shaders. The inverse mapping stage is absent as there is no hardware un-mapping for these setups. We replace this step with texture mapping to generate the final output. It should be noted that though we use texture mapping in these setups, a physical display will not require this, as images will be directly mapped to facets. The overall display resolution is also affected because of this, since the final image is shown using an off-the-shelf projector, of which only about 40% pixels lie on the given shape.

To maintain the resolution quality, as if built using LCD panels, we render all projection based setups to a constant quilt image resolution of 64M pixels (maximum texture size supported on current GPUs).

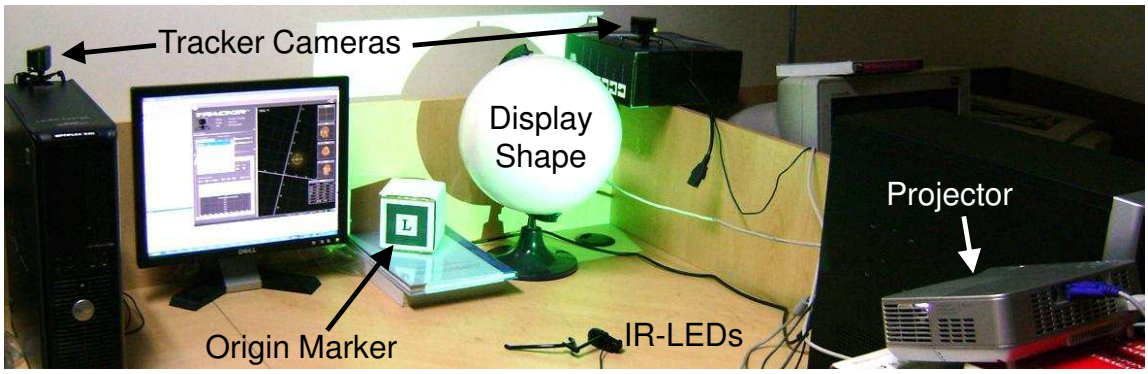


Figure 2.22 Projection based setup hardware configuration

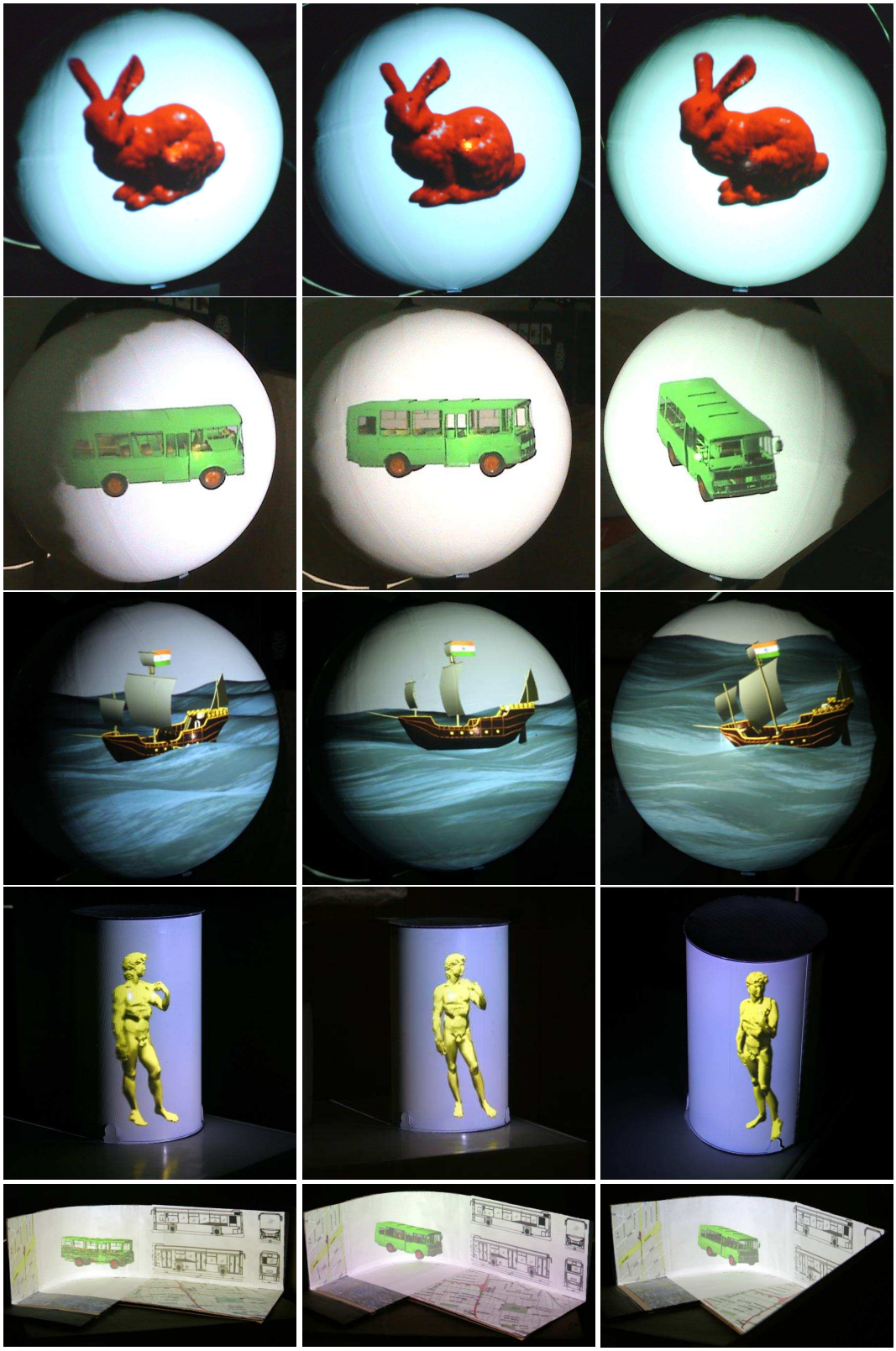


Figure 2.23 Projection Based setups showing various scenes on sphere, cylinder and desktop form factors

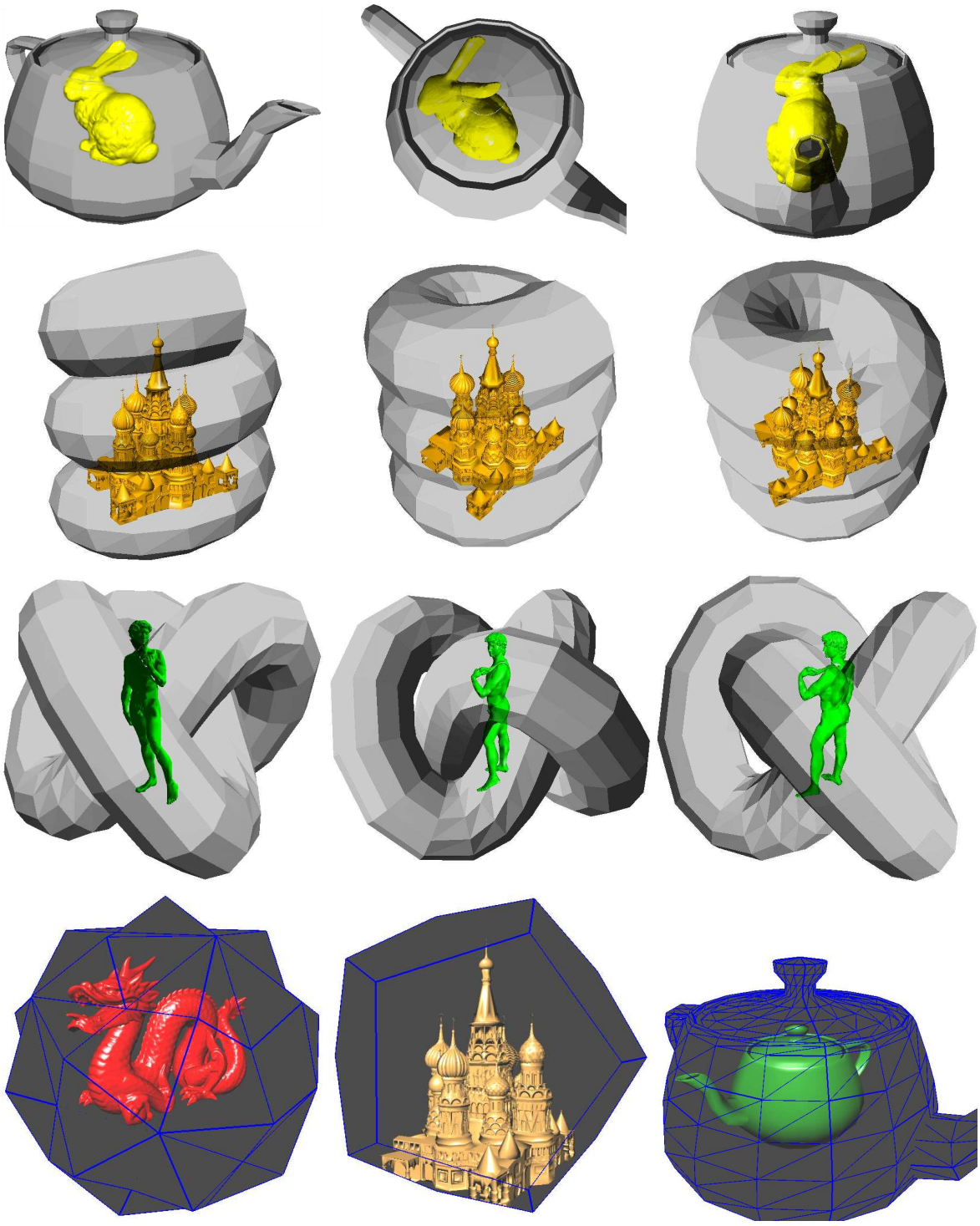


Figure 2.24 Simulated setups showing various scenes on teapot, spring and knot, sphere and other form factors

This can be further improved by changing facet mapping and using more than one rasterization pass. Figure 2.22 shows the configuration we use for various display shapes. These prototypes are intended to demonstrate the scalability of our system to various form factors with large number of display facets and to provide an estimate of system performance if implemented using proper electronics. Figure 2.23 shows projection based setups displaying static and dynamic scenes on spherical (840 facets), cylindrical (216 facets) and desktop (816 facets) form factors.

2.8.3 Simulated Display Setups

We also demonstrate rendering to display surfaces that may be concave or self intersecting using simulated setups. For these setups we render to facets using our pipeline, map the rendered images to the display geometry and observe from the user's perspective. Thus, system performance in terms of rendering time and scalability of the simulated display is a conservative estimate of a real display of the same shape, if built. A 64M pixel quilt image is rendered for each of these setups since the final image is viewed on a low resolution display. Figure 2.24 shows form factors shaped like a teapot (1024 facets), spring (902 facets) and a knot (1200 facets). The system is capable of rendering to these shapes in real time. In the triangle sorting stage, ray-casting finds the facet nearest to the viewer location. This ensures correct depth ordering for facets and allots triangles to the correct facet even when the surface is self intersecting, as is the case in knot and spring.

2.9 Performance Evaluation

In this section we give performance results for different form factors comprising of varying number of facets and orientations along with scene complexity to study the scalability of the framework across different factors. A single Nvidia GTX 580 with 1.5GB of RAM on an Intel Core i7 930 processor with 4GB RAM is used as a testbed for the following experiments. All experiments are reported at 64 mega pixel quilt image resolution with times averaged over a 1000 frame walk-through.

Comparison with Spatial Hierarchies To compare our method with spatial hierarchies, we implemented a dynamic octree structure. The octree is built every frame over the scene triangles on the GPU. Each triangle is assigned to a thread. Each thread finds the octree leaves its triangle intersects spatially and sets it as a part of the VBO of the respective octree leaf nodes. The scene hierarchy is culled to facet frustums in parallel and VBOs of the intersected leaf nodes are rendered to the corresponding facets. We use an octree depth of three, with 512 leaf nodes. Increasing the depth reduces triangles rendered per facet but increases the culling time and hence overall performance is reduced. Dynamic scenes require building the entire structure every frame, which slows down this approach for larger scenes. We found spatial

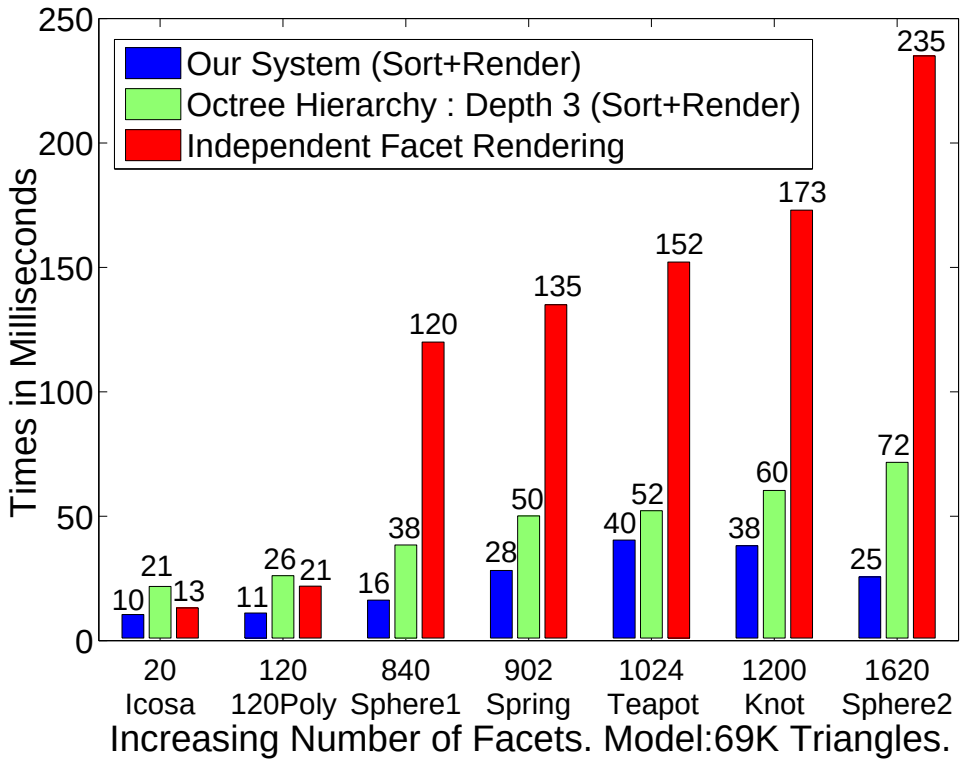


Figure 2.25 Comparing performance of our system with spatial hierarchy and independent rendering for increasing number of facets.

hierarchies to be slower than our method due to the increase in computation as the number of facets increase, as shown in Figure 2.25. The number of triangles rendered per facet increased by a factor of 2 compared to our sorting. Fast sorting on today's GPUs make our method faster than a spatial hierarchy based method, which has more irregular operations that may not map well to the data-parallel hardware.

Figure 2.25 shows an increase in rendering time as the number of facets increase for both octree hierarchy and the independent rendering approach. Our method, however, exhibits uneven per-frame times due to view dependent sorting. The topology of the display surface and visible facets decide the sorting time, which dominates the overall time taken by our system as shown in Figure 2.26. We see that even on a larger display with 1600 facets (sphere2) our method can take less time than on a display with fewer number of facets (teapot, 1024 facets) because of a more even topology of the former. In case of teapot the sorting times vary with the viewpoint when the handle and the nose are in view. This observation can help design better displays. Almost equal number of facets across various views will provide a better performance.

Figure 2.26 gives the time breakup of our system for the same experiment reported in Figure 2.25. We see the sorting time dominates the overall pipeline, more specifically the second pass of sorting takes maximum time. This is because of the expensive triangle-facet overlap tests performed in this step.

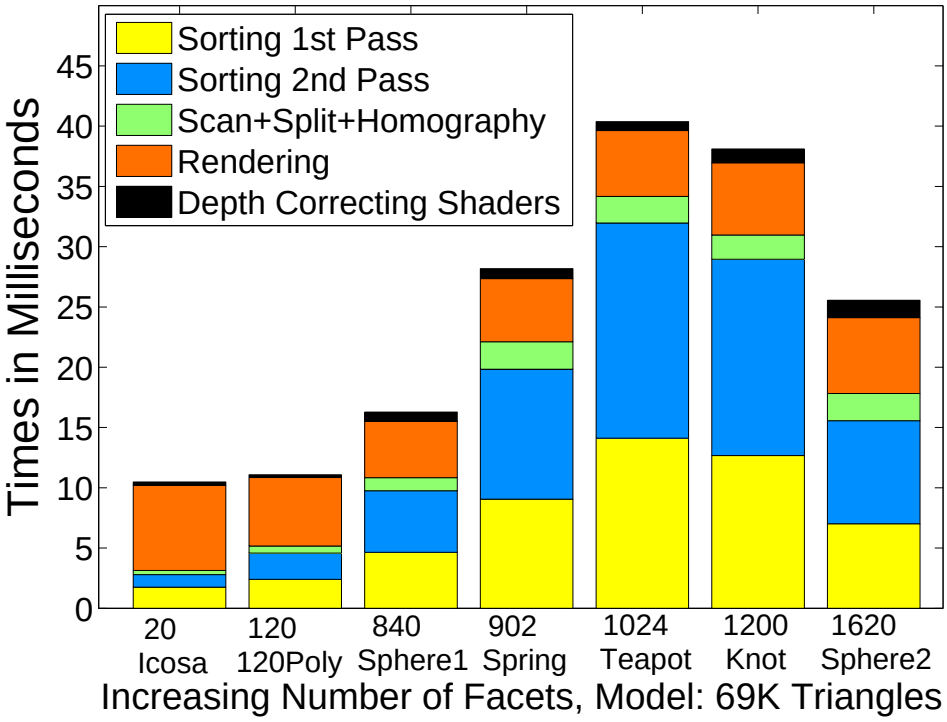


Figure 2.26 Time breakup for our system, showing time taken by each step of our rendering pipeline.

Even though the number of threads used is less, this pass requires projections of facets and triangles and computes intersection of these projections in camera space. The times also vary with the viewer location as more facets and triangles can come in view at various viewpoints. The figure clearly shows triangle sorting to be critical in our scheme. We also note that depth correcting shaders are not expensive as one additional parameter is interpolated by the rasterizer and only one division is computed per pixel. Similarly, the homography computation, transformation of vertices and splitting of triangles to facet VBOs are not limiting factors.

Figure 2.27 examines the rendering time for increasing number of pixels for our method. Rendering depends on facet packing and typically for a closed shape only about 25 to 40% pixels are rendered per frame. The fill rate of current generation GPUs is high and thus even increasing the quilt image size to very large dimensions does not affect the rendering times by much.

In Figure 2.28, we examine the scalability of our system with increasing number of scene triangles (16K–871K) and increasing number of facets (20–1620). Increasing the number of scene triangles increases the number of threads needed for sorting stages. This results in larger sort times and increases the overall rendering time. At 871K triangles we see that the frame rate drops to about 7-10 frames per second. For larger models the sorting load can be distributed to multiple GPUs driving thousands

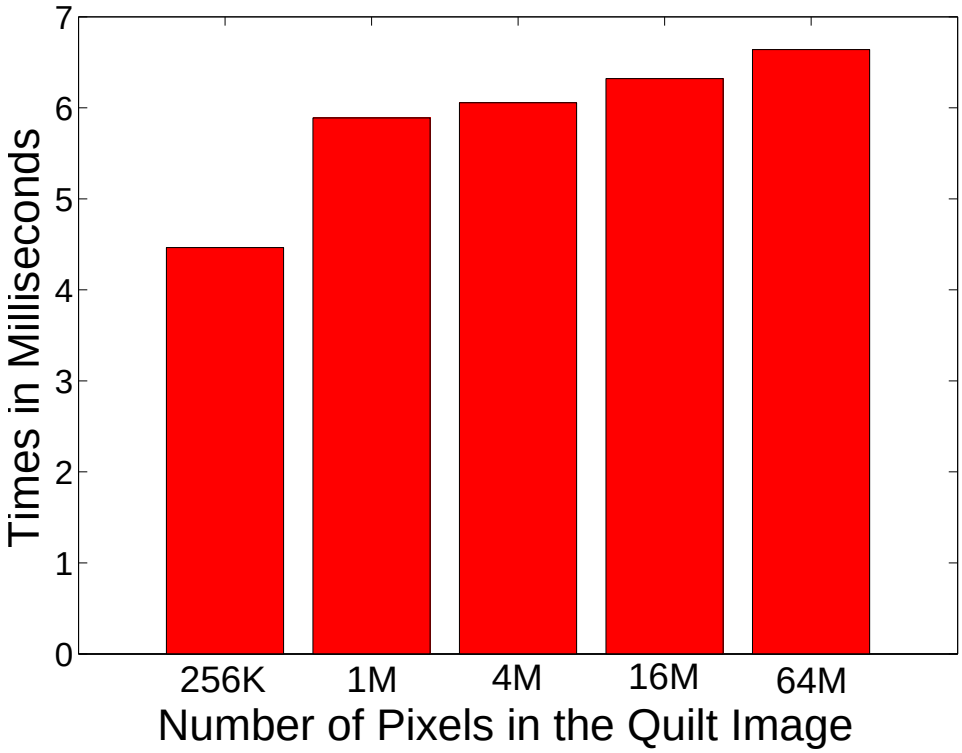


Figure 2.27 Rendering speed w.r.t. increasing quilt image size for a scene consisting of 69K triangles and a display comprising of 1200 facets.

of facets. It can be seen that our system produces real-time frame rates only up to 200K triangles for various display configurations on a single GPU.

2.9.1 Limitations of the Display

The main drawback of our framework is its view dependence. Correct perspective is available only to a single viewer. Others see a distorted image due to a view dependent homography being applied to facet images. This is, however, a common feature of all head-tracked displays. The scene triangle sorting step is the most time-consuming of all steps, especially the second pass. Performance can, however, be improved using multiple GPUs to sort the scene to multiple facets and to generate multiple quilts, as the steps are highly parallel. An implementation of our system requires hardware un-mapping of the quilt image at the display end. A display manufacturer can easily create such a setup using suitable electronics. Our LCD-based system used different VGA outputs instead. Our projected system sacrificed resolution to avoid this un-mapping. Our framework is highly scalable inspite of these issues.

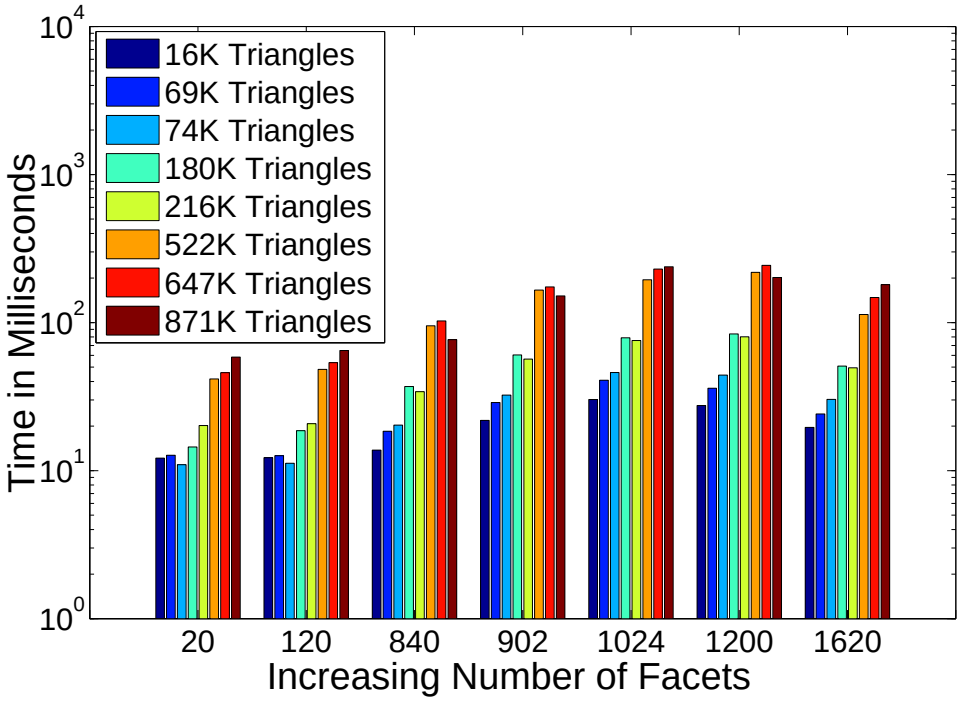


Figure 2.28 Scalability of our system with increasing scene complexity for various display configurations.

2.10 User Study: Utility of a Spherical Walk-around Display

We study user interaction with a spherical display to evaluate the ease of use of a walk-around monoscopic display in this section. Rendering is not evaluated since the display used in this study is a projected setup - which only provides a mock platform for look and feel of our system if implemented using proper electronics. Please see Section 2.5.4 for comparison of various rendering methods. The aim of this study is to see if walk-around displays provide a more natural way to view and interact with virtual objects as compared to flat screens. The focus is on user experience and thus a full implementation of our system is non essential. The spherical shape is chosen for its natural viewing properties. A WiiMote is used to interact with the display, to move the cursor in camera space and also to select objects shown inside the display. We design a simple path-finding task with a hollow connected cube structure with marked start and finish nodes (Figure 2.29). The goal is to find a path between two specially marked nodes through the edges of the cube. Moving around the display is essential to evaluate a path in this exercise. The same experiment is also conducted on a flat display with the same mode of selection input, with additional buttons assigned to move user viewpoint. The rotation angle and the rotation speed are also restricted to simulate walking around scenario on the flat panel. The flat panel is not used with head tracking in this experiment because head tracking will not add anything to the experiment as

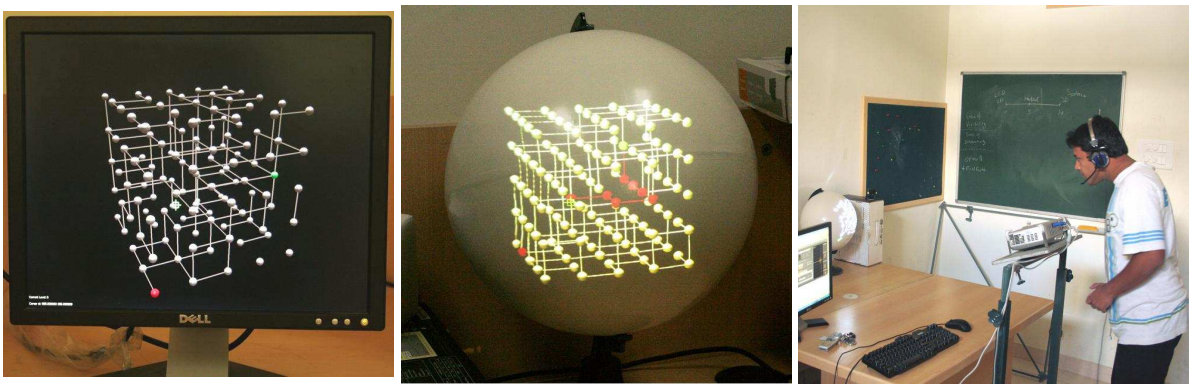


Figure 2.29 Hollow cube structure used for user evaluation. Goal is to find a path from green to red node

the planarity of the flat panel does not allow user view from any angles and will therefore hinder the performance of the flat display.

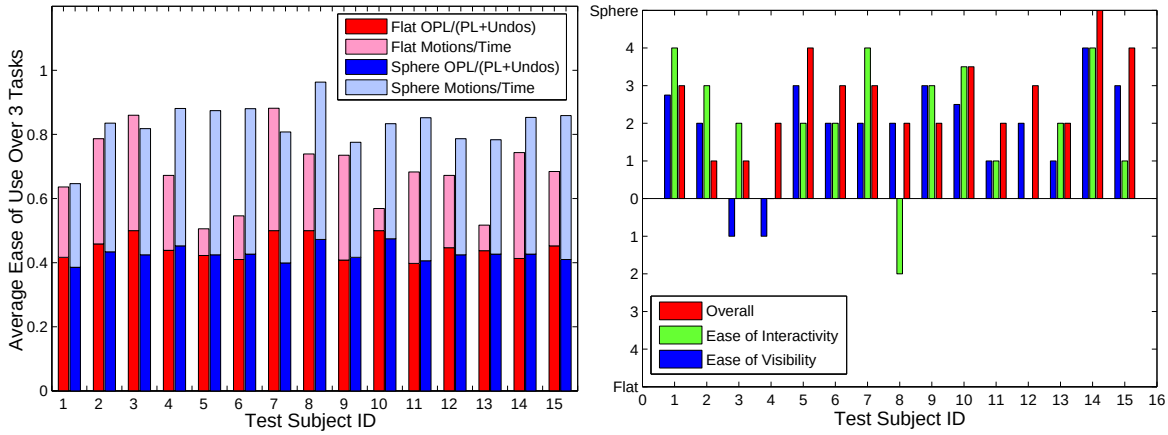
Each user is given the same three tasks to perform on LCD and Sphere with a mock task on each to get familiarized with the display and input modes. We store the selected path length (PL), number of backtracking steps (Undos), viewpoints and the overall time taken to perform the task that are later used to evaluate a subjective measure defining ease of use. A questionnaire comprising three aspects of the experiment is also rated by the user: (i) Ease of visibility, which rates how easy it is to perceive the object/path based on moving around the object on a spherical display as opposed to rotating the object on a flat panel, (ii) Ease of interactivity rates the user input, moving viewpoints using buttons as opposed to physically moving around and (iii) an overall rating which states the user preference.

We define *ease of use* (EOU) based on recorded parameters using a penalizing and rewarding mechanism. Time taken should be penalized along with deviation from the optimum path length (OPL) for each task. Motions help in perception and thus should be rewarded. Backtracking should also be penalized. Considering these we define ease of use as follows

$$EOU = \frac{OPL}{PL + Undos} + \frac{Motions}{Time\ Taken} \tag{2.14}$$

The first term captures the perceptual aspect whereas the second term captures interaction with the display. Both terms are normalized to have a maximum value of 0.5 and are given equal weightage. The metric gives a measure of how easy it is to move around, interact and perceive objects in a display based on our experiment. Fifteen test subjects evaluated EOU on an LCD and on the spherical display. Figure 2.30(a) states their achieved EOU averaged over three tasks.

It can be seen that both flat panel and spherical display show nearly the same deviation from the optimal path length. This is expected as both use monoscopic viewing and are only aided by motion for depth cues. It can also be seen that it is easier to move around the sphere than on the LCD. Motions



(a) Ease of Use for various test subjects.

(b) Subjective user ratings

Figure 2.30 Results of our user study, Figure (a) shows the evaluated ease of use based on recorded parameters and Figure (b) shows the results of the questionnaire.

should also help in depth cues and improve perception. However, because of lower resolution on the sphere, users complained about low quality of viewing, adversely affecting perception on the sphere. Users also indicated that perception might also be affected because of their familiarity with LCD. Figure 2.30(a) favors motion on the sphere for 70% of the users indicating that it is easier to move around a spherical display.

Subjectively, an overwhelming majority of users preferred sphere over LCD as shown in Figure 2.30(b). They indicated it was easier to plan the path on the sphere, and depth variations were much clearer on the sphere. Interaction is also favored suggesting moving around a display to be more comfortable than moving using buttons, also confirmed by the result of Figure 2.30(a).

2.11 Applications

Visualization applications can benefit from quality rendering and interactive frame rates provided by our system. Medical visualization, design prototyping, molecular interactions, etc., require high quality rendering. Our system can provide a glass box interactive display for such applications. More participative games is an area that general polyhedral displays can invigorate, especially walk-around display. With the advent of new motion capture technologies like the Microsoft Kinect, Sony Move and Nintendo Wii, games benefit greatly from novel user interactions. Head tracking is integral to these technologies and hence can drive 3D displays for a single viewer. Pairing this technology with flat displays limits its potential. Figure 2.31 shows 3D interaction with a display shaped like a sphere and a display that fills an entire desktop. A WiiMote is used to shoot the shark and to interactively view a

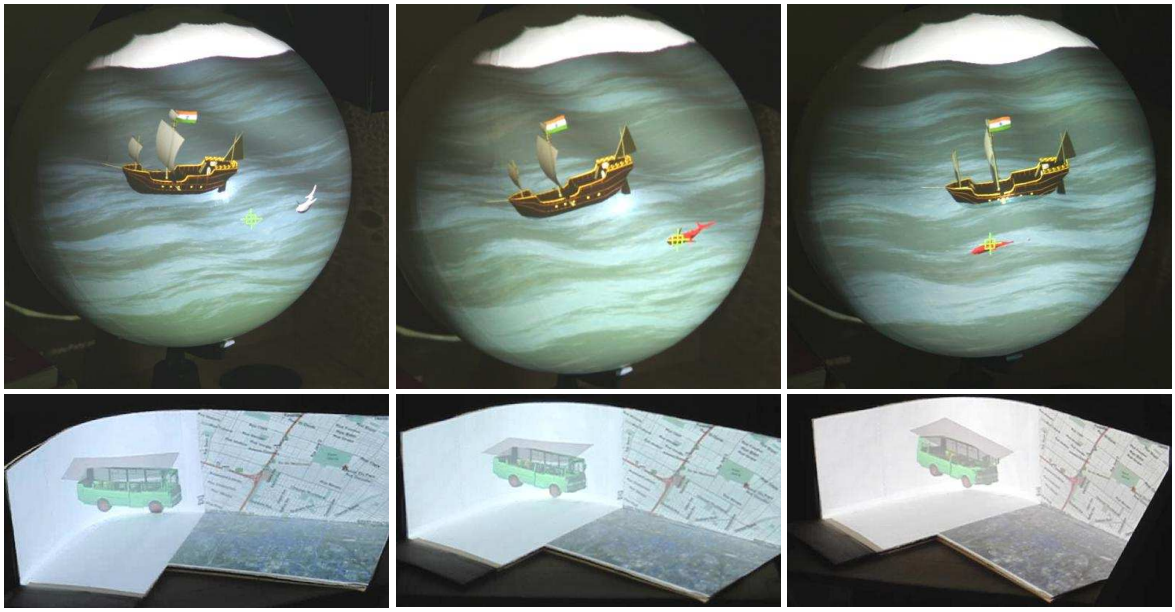


Figure 2.31 Interaction with a spherical and desktop displays

CAD model in these displays. Our user study confirms that such displays to have practical use in real world applications and makes interaction both fun and intuitive.

2.12 Comparison with Projective Texture Mapping

In this section we compare two approaches for generating images for multiplanar displays: the single-pass rendering method we presented and the two-pass Projective Texture Mapping (PTM). We concentrate on the visual quality in the comparison. The PTM approach is as follows.

1. Render the user view to an intermediate image with sufficient resolution to cover all facets.

2. Compute the homography matrix between the user view and each facet.

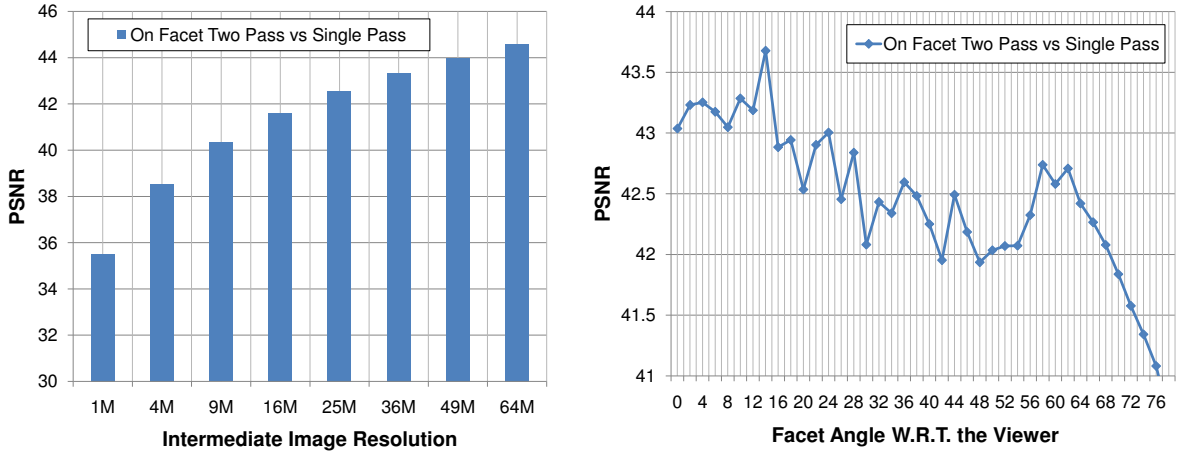
3. Generate texture coordinates for facet vertices by projecting display geometry to the user view.

4. Render each facet polygon independently, setting the homography computed in step 2 as the texture matrix and using texture coordinates from step 3.

We first compare facet images for both methods at 1500×1500 facet resolution, with 1-pass rendering as the ground truth. A cube display and a standard graphics model are used for the comparison. The display geometry is not relevant as the facets are independently rendered. We use the peak signal to

noise ratio (PSNR) defined below as the visual metric.

$$PSNR(I,K) = 20\log\frac{\textit{Maximum Intensity}}{\sqrt{\frac{1}{mn}\sum_{i=0}^{m}\sum_{j=0}^{n}[I_{i,j}-K_{i,j}]^2}} \tag{2.15}$$



(a) Comparing Facet PSNR with increasing intermediate image resolution

(b) PSNR on facet with varying viewing angle, intermediate image: 64M pixels

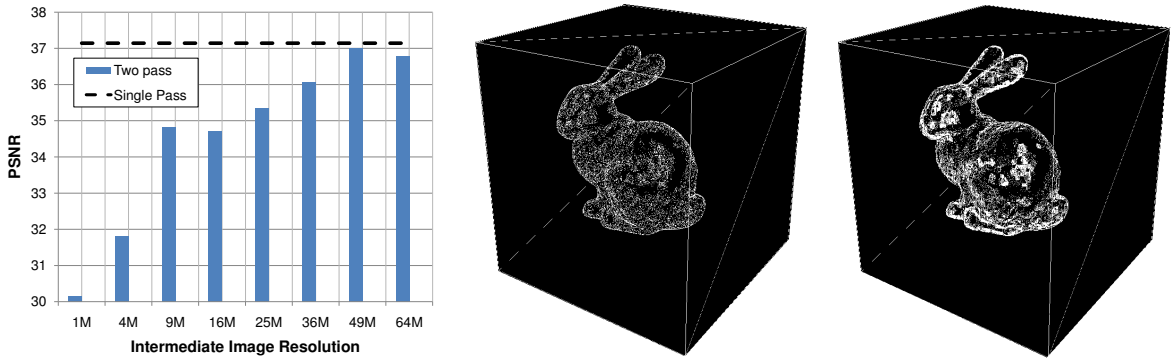
Figure 2.32 Comparison on the facet between the two approaches at facet resolution of 1500×1500 pixels.

Figure 2.32(a) shows the PSNR comparison on the facet for a 250 viewpoint, 360 degree walk-through around the display covering all facets with zooming in and out. The reported PSNR is averaged over the entire walk-through. As seen, PTM cannot achieve quality rendering produced by our 1-pass approach. A maximum of 44.5 PSNR is achieved at 64M intermediate resolution, which is nearly 30 times the size rendered by our 1-pass method.

It can also be argued that facet area projected on the virtual plane decides the number of pixels for that facet in the PTM approach. This can be simulated by changing the facet viewing angle. Figure 2.32(b) compares PSNR between the two methods on a rectangular facet by varying viewing angles. The PSNR decreases with increasing viewing angle, which implies that the PTM method produces worse images at oblique angles.

For visual comparison we also compare the overall display quality achieved from the user's perspective. We compare both methods with a direct rendering of the scene as the ground truth. Another level of interpolation is added when comparing to direct rendering, which reduces quality for both rendering methods and thus lowers the PSNR values. This makes this a weaker comparison than the one taken on the facet itself, since this interpolation is absent in an actual display. The final round of interpolation is however needed for comparison with direct rendering in simulation.

PSNR for each method with respect to the direct rendered image (1280×1024) is shown in Figure 2.33(a) for the same walk-through reported in Figure 2.32(a). Our method generates only as many



(a) Overall PSNR with increasing intermediate image resolution.

(b) Visual difference for single pass rendering

(c) Visual difference for 2-pass method, intermediate image: 36M

Figure 2.33 Comparison of 2-pass PTM and our 1-pass rendering method at 1280×1024 user view. The 1-pass method has a single PSNR as it uses no intermediate image. The model used is bunny and the display is a cube with 1500×1500 facet resolution.

pixels as the number of pixels on visible facets and has only one PSNR value. The quality of PTM improves with the size of the intermediate image. The PSNR of the 2-pass method approaches that of our method only when the intermediate image is 30 times larger. Pixel-by-pixel difference between direct rendering and the two approaches is shown in Figures (2.33(b)-2.33(c)). Both produce errors because of the additional interpolation step. PTM, however, produces more errors even when intermediate image size is 16 times in resolution to that of our method.

Overall, the drop in quality using PTM, both on facet and visually, is significant, especially at oblique viewing angles. The PTM approach performs faster, but we do not recommend its use due to the drop in quality. Quality is most critical for visualization applications and our 1-pass method will produce better results. If speed with limited hardware is more critical as the case may be for interactive games, the PTM method may be used.

2.13 Conclusions and Future Work

We presented a framework to correctly render 3D scenes to multi-planar displays with a large number of facets. Our approach produces correct rendering and maintains interactivity of the application even when the facet counts increases to over a thousand facets. We also demonstrated the scalability of our system with increasing resolution, scene complexity, and number of facets. The framework facilitates rendering to virtually any display configuration as shown in simulation. Practical setups using LCD panels are not hard to build into any shape using appropriate electronics. This can provide a whole new interaction paradigm with the virtual world. With current display technology and advancements

in motion in gaming our framework ideally suits the needs of interactive applications at minimal cost. Users like the additional interactivity of such display over flat panels.

We would like to explore interactions on such displays using touch panels. Such a setup could then enable natural user interfaces to currently challenging problems. For example, with the use of a touch panel spherical walkaround display an artist could sculpt a virtual object as though he was actually working on a real statue. Many 3D displays are about to become practical in the coming years. Our framework combined with these can provide a truly enhanced visual experience of 3D environments and interactivity to the users of the future.

Chapter 3

View Dependent Rendering to Parametric Displays

3.1 Introduction

In this work, we consider displays of arbitrary shape. A surface of arbitrary shape can be covered with display elements or pixels with high density. These can be thought of as being fixed on the surface. We define *parametric display surfaces* as those in which the pixels map to a suitable two-dimensional domain. This provides each pixel with a unique id or index in the parametric space. The display and an image in its target domain are equivalent and we can drive the display by generating the image, which we call its *base texture*. A simple rectangular domain or a subset of it is most convenient for image generation when using modern graphics pipelines.

We present a view dependent scheme to render 3D scenes to general parametric display surfaces. The display surface is specified as a set of equations that define its shape implicitly or parametrically. We assume the mapping of actual display pixels from its base texture is already defined. We, therefore, focus only on rendering to the base texture correctly from the point of view of a user whose head is tracked. Such a display can be thought of as a generalization of fish tank virtual reality display to an arbitrary shape. Our method tessellates the scene based on the geodesic edge length and a user-defined error threshold. We also modify scene vertices, based on per-vertex ray casting, such that the final image appears correct to a user's viewpoint. The ray-surface intersection procedure, geodesic length computation and 2D image mapping are assumed to be known for the given surface. We exploit the tessellation hardware of the SM 5.0 GPUs to perform the error checking, polygon splitting, and rendering in a single pass. This brings the performance of our approach closer to rasterization schemes, without needing ray tracing. We demonstrate our rendering scheme using a few real curved surfaces of algebraic form, namely, sphere and cylinder. We also show more complex algebraic display surfaces using simulations. We achieve over 100 FPS on a scene with close to a million triangles when rendered to a spherical parametric display.

3.2 Related Work

Curved surfaces have been used to display 2D images using texture mapping. Non-planar arrangements of displays have also been explored, an example being the fish eye view generated using multiple projectors projecting onto a dome using texture mapping [90], [27]. Benko et al. [20] extend texture mapping to curved surfaces in order show and interact with view dependent 3D content. Bimber et al. [24] extended the two pass texture mapping approach to any surface using stereoscopic projection. Extending this approach to display 3D scenes is not trivial. Image space methods have also been explored that auto-calibrate multiple projectors seamlessly on to an unknown surfaces, a survey of such techniques is discussed in [29].

Fish tank virtual reality allows realistic exploration of 3D scenes using head tracking. Each planar facet of the display shows an image based on the viewer location such that the combined effect produces the illusion of virtual objects placed in viewer space or vice-versa. Though the notion of FTVR displays is old [43], one of the first implementations was the CAVE virtual environment [36]. Five back-lit planes were projected on to create a virtual enclosure. Off axis rendering [38] was used to generate projection matrices through which the scene was rendered independently for each plane. FTVR displays have followed this template with alterations to the basic framework, please see Section 2.2 for various prototypes developed based on this scheme. In our earlier work [50] we also extend the FTVR display framework using a GPU algorithm for better image generation for multi-planar display surfaces that can approximate arbitrary shapes. The present work focuses on rendering to a display that is exactly defined parametrically.

3.3 As a Computational Display

Our rendering scheme follows the Computational Display framework given in Figure 1.4. The difference being instead of using multiple conventional displays to show the final result a single display of a known parametric shape is used, as shown in Figure 3.1. The sub-image generation does not work for conventional displays in this scheme, but is a parallel per-vertex compensation for the display curvature. A single base texture image is generated per frame which is mapped onto the display using fixed texture coordinates. Prior knowledge about the display surface - its parametric equation and base texture mapping - are used to achieve this. Triangles are broken on the fly and their vertices are adjusted to compensate for the display geometry based on the geodesic length of each triangle edge on the surface as viewed from the user's point of view.

3.3.1 Physical Processes Involved

The physical processes involved in our rendering scheme are the tracking of the viewer's head position, location of the display with respect to the viewer and the display surface equation along with the base texture mapping. These parameters are used to render to any parametric display correctly.

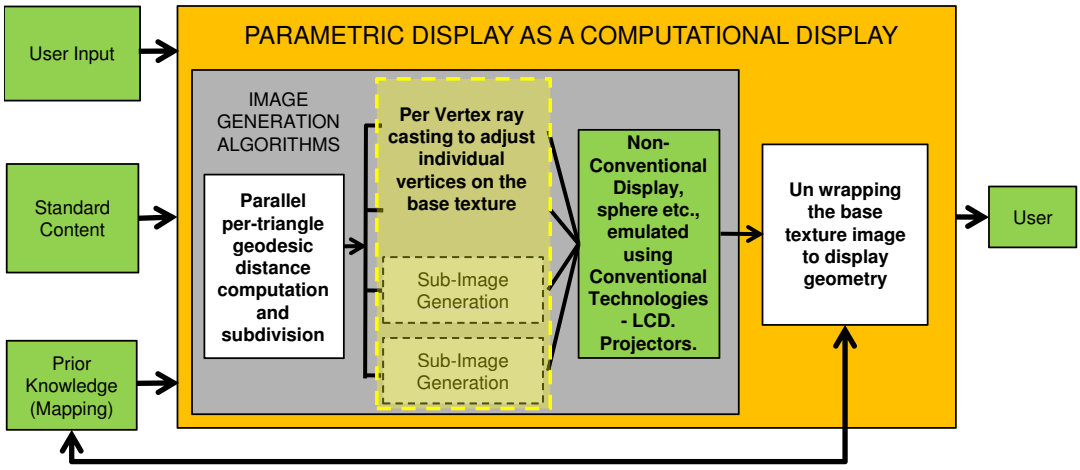


Figure 3.1 Parametric Display as a Computational Display.

We assume the geodesic length computation and ray-surface intersection procedures are also known. The geodesic length computation is used to break large triangles to a user specific error threshold and the ray-surface intersection procedure is used to correct vertex coordinates to produce a perspectively correct view on the surface.

3.3.2 Algorithmic Load Distribution

In the case of parametric displays, the load distribution is entirely parallel. Each triangle is processed in parallel using the shader model 5.0 tessellation control shader and its vertices are later also processed in parallel using the tessellation evaluation shader (described in Section 3.4). The CPU updates the user location to the GPU. The surface geometry is known ahead of time, based on which triangle splitting and vertex updation can take place using per triangle and per vertex shader codes. Thus, the image generation process follows the computational display frame work in terms of algorithmic load. The view, however, is generated on a single display as opposed to a montage of conventional displays.

3.4 Shader Model 5.0 Tessellation

In this section we give a brief introduction to the Tessellation capability of the DirectX 11, SM 5.0 graphics pipeline since the conventions are used later in our rendering pipeline. With the introduction of SM 5.0 a new hardware unit is added in the graphics pipeline, called a Tessellator. The purpose of which is to produce and manipulate primitives on the fly, i.e. to generate more primitives given an input primitive. For triangled scenes this yields more detail given a relatively low input mesh model. This can

be used to implement varying functionality such as bump mapping and dynamic level of detail (LOD). As the user moves closer to an object triangles could be increased and modified to generate greater detail and similarly decreased when the user moves away.

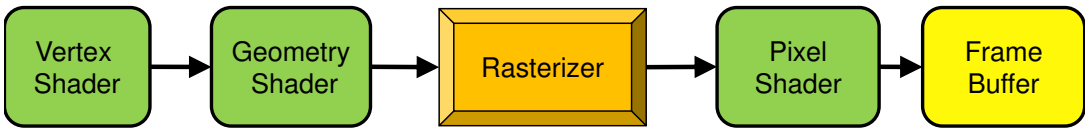


Figure 3.2 The Shader Model 4.0 programming pipeline. All green ovals are programmable units. Note only Rasterizer is a non-programmable unit in SM 4.0

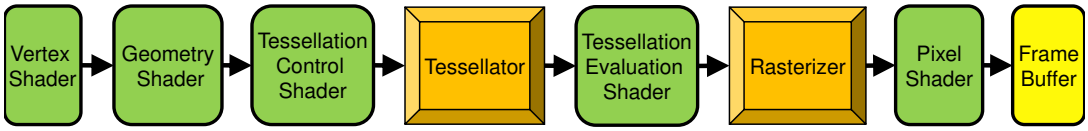
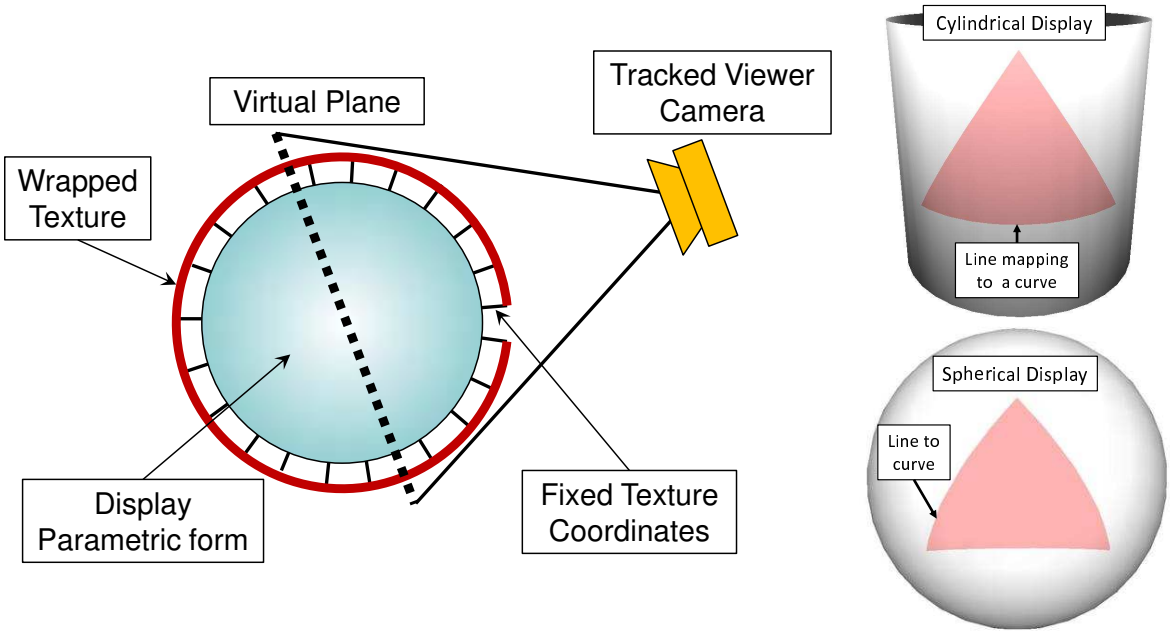


Figure 3.3 The Shader Model 5.0 programming pipeline. Green ovals indicate the programmable units. Two more programmable units are added in SM 5.0 along with another non-programmable unit.

The SM 5.0 pipeline (Figure 3.3) differs from the previous generation hardware (Figure 3.2) in terms of programming capabilities. The new pipeline introduces two new programmable stages, one before and one after the new hardware tessellation unit. These *shader units* control the input and manipulate the output of the tessellator. The Tessellation Control Shader (called the Hull Shader in DirectX 11) is used to specify the input parameters per primitive entering the tessellator. For example, for triangles, the tessellation control shader requires the breaking parameters - which define how the triangle must be broken in order to create more triangles. This requires three parameters per edge, called the *TesslevelOuter* parameters that define in how many parts a triangle edge will be broken (Figure 3.5(a)(top)). Another single parameter called the *TessLevelInner* is specified to tell the tessellator how to break the inside of a triangle (Figure 3.5(a)(bottom)). The inside of a triangle is divided based on a fixed pattern of triangles and is non-programmable. Based in these parameters the tessellator generates a new set of triangles in the same plane as the original input triangle. The vertices of the newly generated triangles are mapped in [0,1] range, and based on the input vertices can be scaled to respective coordinates.

The second shader, the Tessellation Evaluation Shader (called the Domain Shader in DirectX 11) is used to manipulate the newly generated triangles. Since the newly generated vertices are in [0,1] range this shader requires to map these to world coordinates. This is done by acting on each newly generated primitive in parallel. This shader can apply any desired transformation on newly formed vertices to produce various effects such as bump mapping, smoothing, etc. We use this shader to warp newly formed vertices on the basis of display geometry.



(a) Display parametric form, texture image and viewer camera in the same coordinate frame

(b) Lines mapping to curves

Figure 3.4 Parametric display form along with texture wrapped around the display, note line on the texture mapping to a curve on the display surface.

3.5 Rendering on to Parametric Display Surfaces

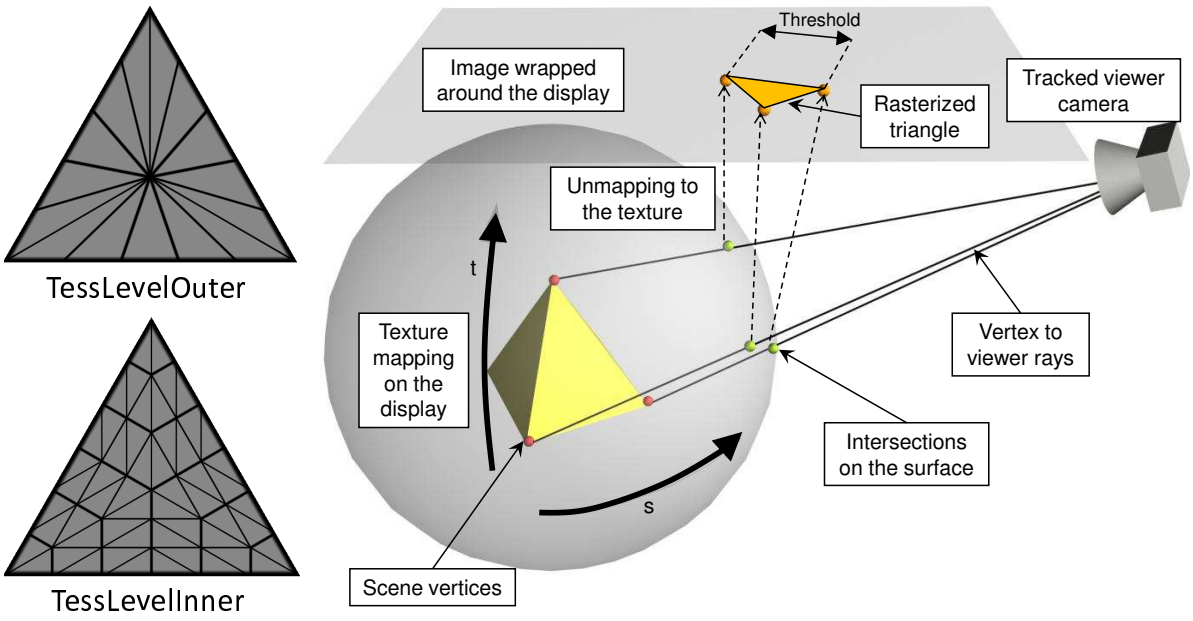
Rendering to a parametric display surface is equivalent to rendering to its base texture, which is wrapped around the surface of the display (Figure 3.4(a)). This texture is the target for our rendering pipeline. Given a curved display shape, it can be seen that lines on the texture can become curves (Figure 3.4(b)). The position of end vertices may also change due to the mapping process.

Ray tracing the scene to visible pixels is the natural way to render to parametric display surfaces. This produces correct results at heavy costs. Our rendering scheme approximates this using rasterization and tessellation to a user-specifiable error bound. We modify the scene to be displayed so that a perspectively correct view is seen from the observer's viewpoint. Two attributes must be preserved to do this on an arbitrary surface: Linearity and vertex location in the viewer's eye coordinates must be preserved.

3.5.1 Triangle Division to Preserve Linearity

In order to preserve linearity on the surface, lines in the scene should be pre-warped in the texture image, to compensate for the local surface curvature. Correct solution to this problem requires non-linear rasterization of the scene to the texture. It can be observed that the curvature effects are negligible for small line segments in the world. In the limit case, ray tracing to base texture pixels will provide correct results. We approximate non-linear rasterization by subdividing or tessellating large triangles

on the fly. This breaks line segments into smaller ones. We exploit the tessellation unit available in the Shader Model 5.0 GPUs to do this. The subdivision and rendering are preformed in a single pass. For a given triangle three TessLevelOuter values are passed to the tessellation hardware, one for each edge. The tessellator divides each edge into equal size segments using these values and forms smaller triangles as shown in Figure 3.5(a), top.



(a) Tessellation Levels (b) Finding intersection on the surface along with the corresponding pixel value

Figure 3.5 Tessellation levels, and finding edge length and vertex location based on per vertex ray casting.

The length of a line segment to be drawn is fixed based on the local display geometry. To do this, we intersect the ray from the viewpoint to each triangle vertex with the exact display surface. We assume a ray intersection solution is available for the given shape. Ray tracing methods using shaders are now available for higher order surfaces [108]. The geodesic length of the triangle sides determine the level of subdivision so that a user-specified error is not exceeded. Geodesic length computation depends on the surface used and may differ for various shapes. We use the great circle distance for our spherical prototype. The tessellation parameters are computed adaptively based on the triangle edge lengths and the error threshold. Figure 3.5(a)(top) shows that even after dividing the edges into smaller segments the newly formed triangles may end up with edges greater than the threshold. We set TessLevelInner as the maximum of TessLevelOuter values to ensure that all triangles will have edge lengths less than or equal to the threshold, as shown in Figure 3.5(a)(bottom). The triangle tessellation control shader is shown in Algorithm 5.

Algorithm 5 Tessellation Control Shader

1: Input: Vertices[3], Threshold
2: Compute geodesic length of each edge on the surface as D[1 to 3] based on ray-casting.
3: TessLevelOuter[1 to 3] = D[1 to 3]/Threshold
4: TessLevelInner = max(TessLevelOuter[1 to 3])
5: Output: TessLevelOuter[3], TessLevelInner

3.5.2 Modifying Scene Vertices

Subdivision of the triangles is not enough to ensure correct rendering. The resulting vertices should be moved to ensure an undistorted image. Modified positions depends on the local display surface and the viewer location. We compute new vertex positions using per vertex ray-casting, finding intersection of each eye-vertex ray with the display surface and changing the vertex coordinates to the new location. The vertices can then be mapped to the texture space using the mapping from display surface to the base texture for rasterization, as shown in Figure 3.5(b).

Algorithm 6 Tessellation Evaluation Shader

1: Input: Vertex, Viewer location, Viewer camera
2: Compute the ray from viewer head position to the vertex
3: Compute ray-surface intersection using closed form solutions for simple algebric surfaces or using interval based methods for implicit surfaces [108]
4: Convert the intersection point (x, y, z) to (s, t) in $[0, 1]$ range by unwrapping the coordinates using the given texture mapping.
5: Compute depth per vertex by projecting it through the viewer camera.
6: Output: Vertex

We move the newly formed vertices in the normalized canonical space. The tessellation evaluation shader is used to generate vertex coordinates in this space. Each newly formed vertex finds a ray to the viewer location. The ray is intersected with the surface and the intersection point is converted to a (s, t) coordinate in $[0, 1]$ range using the base texture mapping. The vertex then sets its (x, y) coordinate to the new (s, t) location while the depth value is computed using the viewer camera. The tessellation evaluation shader is given in Algorithm 6. Once all the vertices have been computed for a primiive the primitive is then rasterized to produce the texture image as shown in Figure 3.5(b).

3.5.3 The Overall Rendering Pipeline

The overall rendering pipeline (Figure 3.6) works in a single pass to render a given scene onto the parametric display. The pipeline maps to the SM 5.0 pipeline (Figure 3.3) by replacing tessellation control shader and tessellation evaluation shaders for the oval boxes. A head tracker sends the viewer location to the system. The surface shape and the mapping to the base texture are known to the shaders. Triangles are subdivided and their vertices are modified by the shaders. The resulting mesh is then sent

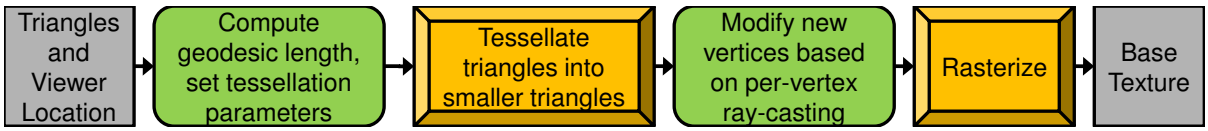
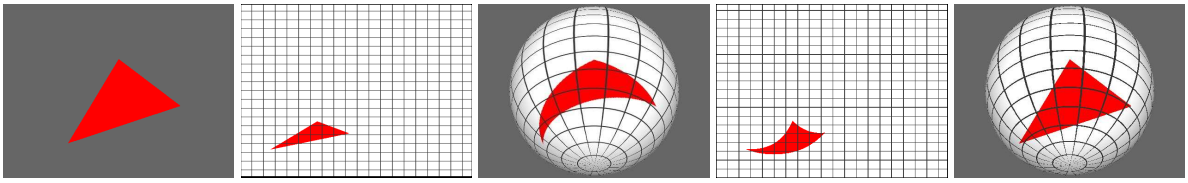


Figure 3.6 Rendering pipeline for view dependent parametric display surfaces.

down to the rasterizer to generate the base texture image which is mapped on to the display surface using fixed texture coordinates. The image when wrapped around the display appears perspectively correct from the viewer's point of view. Figure 3.7 shows the 3D scene, the rendered base texture image with and without tessellation and the view on the surface as observed from the viewer's point of view.



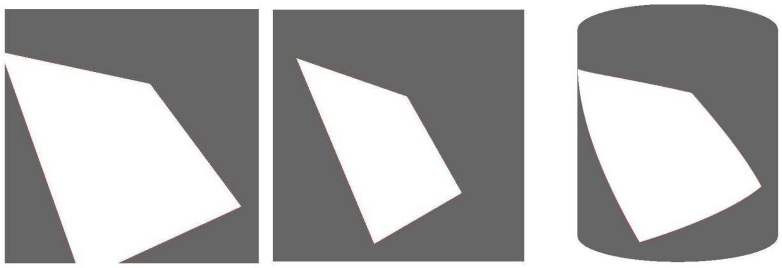
(a) (b) (c) (d) (e)

Figure 3.7 Tessellation and ray casting to generate inverse curve on the texture. Figure (a) the scene, (b) texture without tessellation, (c) non-tessellated texture on display, (d) texture using tessellation and (e) tessellated texture on display.

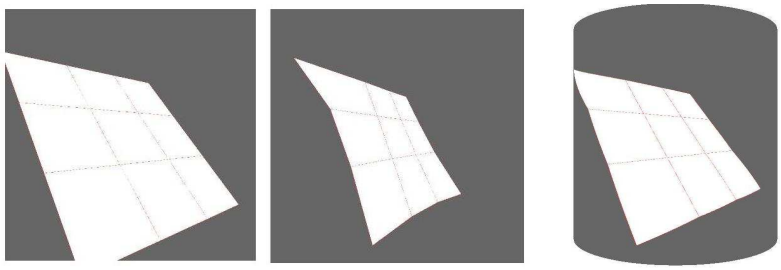
The effect of amount of tessellation is explore in Figure 3.8. A plane is tessellated and mapped on to a cylindrical display using varying amount of tessellation. The degree of tessellation directly translates to better image quality on the surface. This however, is only up to a certain limit, beyond which tessellation does not add further quality gain as shown in our experiments (Figure 3.12(b)). This limit is what the user specifies as the threshold value. For complex scenes with lot of triangles the threshold may never be reached and hence the our pipeline does not tessellate unnecessarily, providing quality renderings at high speeds.

3.6 Display Prototypes

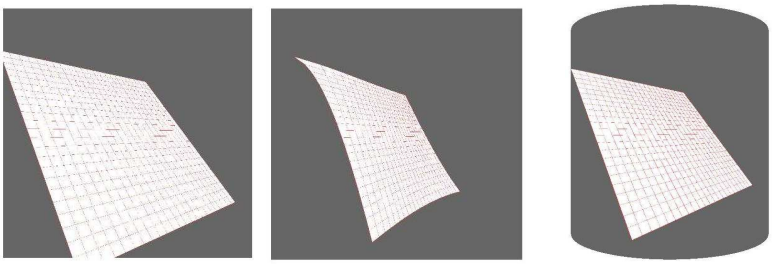
We demonstrate our rendering mechanism using projected and simulated setups. Projected setups allow only a limited number of pixels on the surface, but still demonstrate the entire process. Our prototypes show $30 - 40\%$ of the projected pixels on the surface. The base texture is wrapped around a virtual replica of the physical display and is projected on to the calibrated physical display using a calibrated projector. For the simulated displays, we use a base texture resolution of 64M pixels to support many types and sizes of displays. The image is wrapped around a graphics model of the display and is observed from the user's perspective. Head is tracked using two infrared head trackers (TrackIR5)



(a) Scene with no Tessellation. Left: Original Scene, Center: Unwrapped Image, Right: Texture on the Surface



(b) Scene with low Tessellation. Left: Original Scene, Center: Unwrapped Image, Right: Texture on the Surface



(c) Scene with high Tessellation. Left: Original Scene, Center: Unwrapped Image, Right: Texture on the Surface

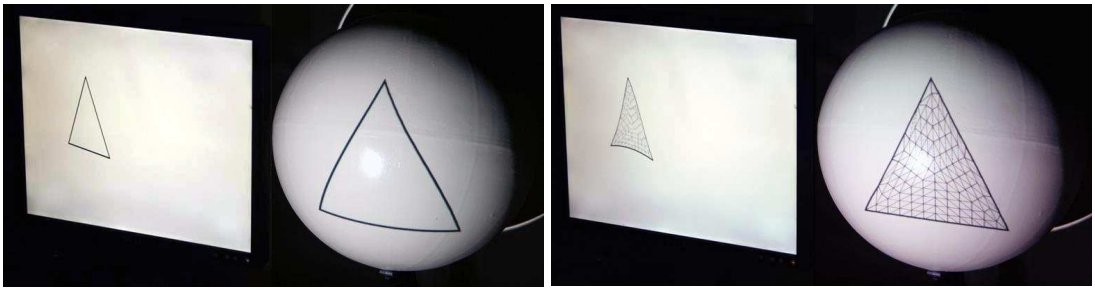
Figure 3.8 Effect of various degree of tessellation for a scene consisting of a plane on a cylindrical parametric surface. Note how increasing the tessellation level produces better image on the surface when observed from the viewer's perspective.

for the real display, with a refresh rate of 120Hz the latency is minimized. Figure 3.9 shows various scenes in spherical and cylindrical prototypes. A single GTX470 GPU is used to tessellate and render the scene.

In Figure 3.10 we show the effect of tessellation on a real display. Lines map to curves when projected on to a spherical display without tessellation, even when vertex positions are corrected based on viewer location. Edges of the triangles become straight with tessellation enabled along with vertex position correction from the point of view of the observer.

The display shown in Figure 3.11 has parametric form $y = x^2 + z^2$, and a high curvature. Our rendering pipeline is capable of handling surfaces with high curvature. The curvature of the surface is compensated for by using a smaller distance threshold for tessellation (1.2% of the height of the

Figure 3.9 Projected display prototypes showing various scenes.



(a) Without Tessellation (b) With Tessellation

Figure 3.10 Triangle rendering with and without tessellation to a spherical display. LCD shows the texture image wrapped around the display.

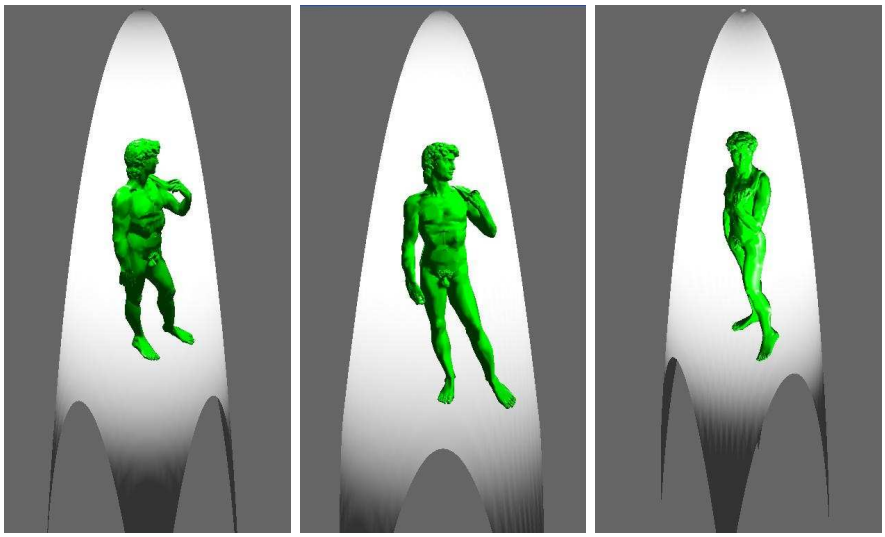


Figure 3.11 Simulated display, surface equation $y = x^2 + z^2$.

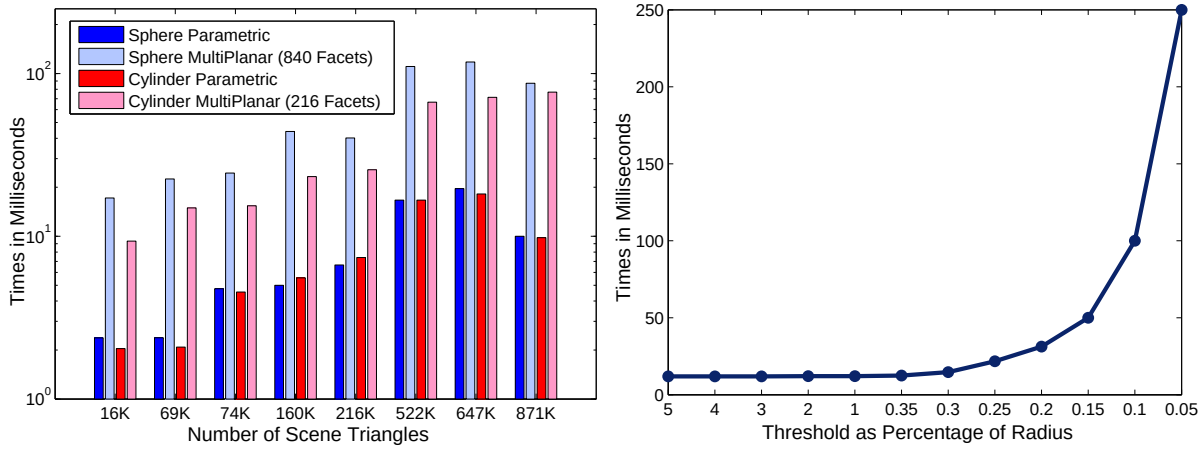
display). Decreasing the threshold to small percentages has minimal affect on performance as reported in Section 3.7.

3.7 Performance Evaluation

We demonstrate the scalability and performance comparison of our rendering pipeline in this section. We use a single Nvidia GTX 470 GPU with 1.2GB of RAM on an Intel Q6600 processor with 2GB RAM as our testbed for all the experiments reported in this section.

Figure 3.12(a) compares the rendering times of our method with our previous approach, approximating the surface using multiple planes [50], for varying scene complexity. Typically we see an increase in rendering times with increasing number of scene triangles. However, the reverse may also happen: the scene with 871K triangles shows faster rendering as compared to scenes with lesser number of triangles. This happens because a scene with a large number of triangles will have smaller triangles, such lie within the threshold range and are thus not tessellated. It can also be seen our method is orders faster than [50] because of single pass rendering. We see an average speed gain of $20\times$ over our previous method.

Figure 3.12(b) explores the effect of varying tessellation threshold for the 817K triangle scene on the spherical display. It can be seen that even at distances as low as 1% of the radius of the sphere, the performance penality due to tessellation is negligible. Performance degrades after 0.25%, which is a very small distance as compared to the size of the display. Visually no variation beyond 1% of the radius as tessellation threshold was observed for any scene.



(a) Rendering times with varying scene complexity compared to [50]. Threshold = $0.01\times$radius.

(b) Effect of reducing the threshold. Display: Sphere, Model: Dragon (871K triangles).

Figure 3.12 Performance analysis of our rendering pipeline

3.8 Conclusion and Future Work

We presented an approximate high performing view-dependent rendering scheme for parametric surfaces in this chapter. With motion capture technologies become more prevalent, our scheme provides an ideal match for walk-around gaming applications. Our prototypes demonstrated the look and feel of such displays. The method, however, is only applicable to surfaces that can be ray-casted. More general surfaces are now finding ray-casting solutions [108] and can become suitable display candidates.We would like to explore rendering to more complex display shapes. Surfaces defined by splines can be used to approximate any surface and would provide a general solution to any display shape.

Chapter 4

Increasing Intensity Resolution on a Single Display

4.1 Introduction

High dynamic range images are commonplace today as even ordinary cameras are capable of capturing 12 or more bits per channel. However, displays to view such images are not easily available; most displays are limited to a low color resolution, typically 8 bits per channel. Though much research is going on in high dynamic range (HDR) imaging, displays to show such images are still far from being commonplace. Most techniques to render HDR images to low-bit displays involve image enhancement using tone mapping operators. Images rendered using these methods necessarily lose information in the process, though they are visually appealing. Retaining all intensity details calls for displaying in-between-intensity levels. Visualization of medical, scientific, and astronomical data are among the applications that can benefit from such displays. Regular images too can benefit from a mapping that shows more intensity levels as more detail is presented to the viewer.

Spatial resolution of a display has grown over the past few years even though the color resolution has not. Vertical refresh rate has also increased well beyond the limit dictated by the human visual system. In this chapter, we present a method to add more perceivable intensities on everyday displays by exploiting their increased spatial and frequency characteristics. We mix pixels spatially and temporally on a single display to generate in-between-intensities not present on the base display. The additional facility sacrifices spatial resolution, refresh frequency, or both in order to gain intensity resolution. We can increase the intensity resolution without loss in human perception by switching between different intensities at each pixel. For example, on a display capable of 120 Hz vertical refresh, two intensities can be mixed to produce a new intensity that is the average of both, by reducing the effective refresh rate to 60 Hz. A similar argument can be made for spatial mixing by treating a group of, say, 2×2 pixels as a high-resolution pixel. This can be done on ordinary displays without hardware modifications, providing a low-cost way to display more intensity/color levels.

Our method differs significantly from other methods that map HDR images to low-bit displays such as tone mapping, which aim to maintain the visual feel without changing the intensity scale. We increase intensity resolution of the display at lower refresh rates or spatial resolution. This enables a viewer to

see the in-between-intensities not present in the base display and make distinctions based on them. Our methods are built on the ideas of dithering and half-toning techniques which are typically used for the static print medium. We extend these techniques to an inherently dynamic computer display taking advantage of its ability to change across time.

We show results using high frequency CRT displays and evaluate our system both quantitatively and qualitatively. Quantitatively, we evaluate the method using a high-bit camera sensor which measures the actual intensities on the display to verify if more are seen. We also evaluate our system subjectively, using human subjects. A user study evaluates our technique over various intensity scales, the goal of which is to visibly distinguish in-between-intensities between the two adjacent levels of a basic 8-bit display.

4.2 Related Work

Capturing and generating high dynamic range content is commonplace in today's world. Debevec et al. presented a way to capture HDR images using conventional cameras by taking multiple exposures of the scene [37]. In many domains such as medical, astronomical and visualization, the data produced is of high dynamic range. Games simulate HDR using glow and bloom effects [57]. Real-time HDR texture mapping is also shown for computer graphics [33]. A high dynamic range display is thus needed to provide high color resolution required by these applications. Physical methods such as limiting the light scatter using per-pixel reflectors have been shown. These, however, are hard to scale with the number of intensities. High dynamic range can also be achieved using multiple lower-bit images overlaid on top of each other. Seetzen et al. prototype two HDR displays using this approach, one using a projector to back-project on an LCD and another using an array of LEDs behind an LCD [105]. Such a system can provide one additional bit to the display. Multiple projectors can also be used to produce a similar effect. By aligning four projectors pixel-by-pixel, one can add two additional bits to the color resolution of a single projector [102]. These methods, however, cannot scale beyond a few bits as the required number of physical devices keep on doubling.

Directly viewing high dynamic range data on a low dynamic range display is cumbersome. A windowing method is usually employed to select the ``best'' high range intensities to map to lower bit displays. An alternate approach is to map the high color range to a lower range using local contrast adjustments or tone mapping [61, 60]. This produces pleasing images to a user as the images are adjusted according to the human visual system and are thus more appealing to a human observer. Though visually appealing, these methods do not add more intensities to the display. A comparison of this method to an HDR display is presented in [66].

Image enhancement algorithms such as dithering and half-toning rely on human perception to produce intensities over spatial integration of nearby pixels. These are capable of producing better images by propagating quantization errors to neighboring pixels. The methods are typically used to produce visually acceptable images from low-bit inputs in print. Half-toning integrates pixels spatially in two

ways using amplitude or frequency modulation [31]. In amplitude modulation, the location of error diffused pixels is static, the intensity is varied by changing the pixel size. Frequency modulation keeps the same pixel size but varies its frequency across the region to produce dense and sparse intensities. We extend these methods to a dynamic display using spatial and temporal integration to add more intensity levels on display than physically available on it.

4.3 As a Computational Display

Our framework, as stated, extends the amplitude and frequency modulation methods of dithering and half-toning to active computer displays. The output, hence, is on a single conventional display as opposed to multiple displays. The idea is to show HDR images captured using conventional/unconventional means on an off-the-shelf display. This is achieved by exploiting two imaging phenomenon - the human visual system and the display model. Our design agrees with the computational display framework (Figure 1.4) since we enhance the display capabilities of a base display using prior knowledge about these physical phenomenon (Figure 4.3). The images shown on our displays are standard content, consisting of 10 or 11 bit HDR data. These are broken down into multiple lower resolution sub-images that may be shown spatially or temporally to produce a close approximation of the high resolution image in the observer's eye. The sub-image generation methods we present follow serial (for images) and parallel (for videos) implementations as needed based on the HDR input type. Pre computation of HDR data to low intensity data is performed off line and can also be done serially or parallely as shown. This is a pre-processing step and does not restrict the performance of our system.

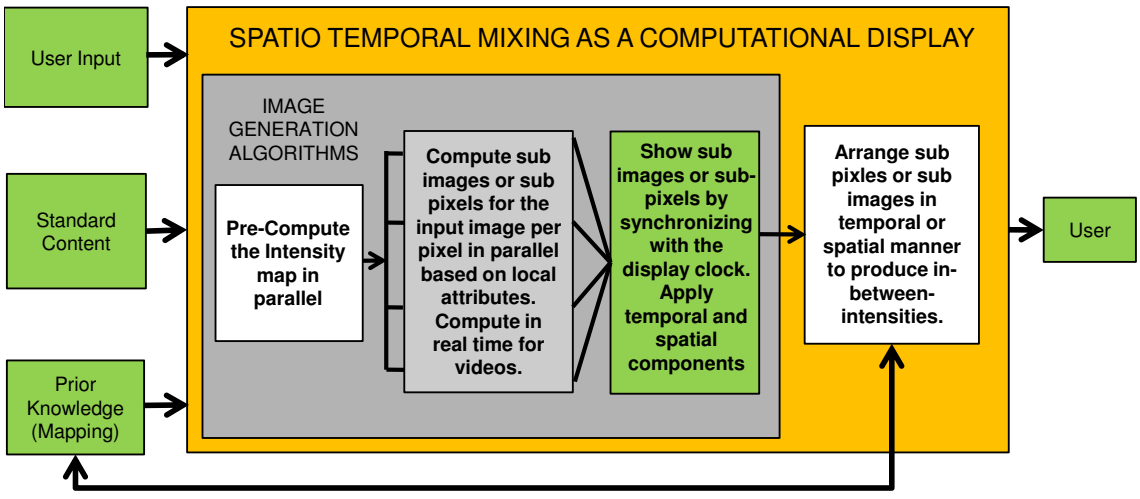


Figure 4.1 Spatio Temporal Mixing as a Computational Display

4.3.1 Physical Processes Involved

Two physical processes are involved in our mapping of HDR data on to conventional displays. The first involves the response of the eye with spatially and temporally varying intensities. The second on how these intensities are generated by the display itself. We use both these processes in all our mapping methods to intelligently map high bit intensities to lower bit intensities. In our design both temporally and spatially varying low bit intensities (or combination of these) are used to generate the perception of the input high bit intensity in the observer's eye.

4.3.2 Algorithmic Load Distribution

The load distribution of our methods are relatively simple as we process each high-bit pixel individually. This maps to a parallel design directly. We break the input high-bit intensity to lower bit intensities as a pre processing step. Input high-bit image is then decomposed into a set of sub-images. Based in this decomposition we can process each input image pixel in parallel or serial manner. For videos a parallel implementation produces the best performance.

4.4 Background

In this section, we introduce the display and the human visual system models we base our intensity expansion techniques upon. Both are elaborate subjects in themselves. Here we only focus on the aspects relevant to our system. Although our methods can be applied to any display, we focus on CRT displays since they provide the high refresh rates needed for our system.

Human Visual System: The human visual system (HVS) is a complex machine. In the context of this work, our interest lies in the description of perceived intensities over spatial and temporal arrangements of pixels. From a distance, a spatial arrangement of pixels produces an average intensity in the viewer's eye as the eye cannot distinguish between the individual intensities [98]. The critical diameter defines the spatial area over which intensity summation occurs. Intensity summation occurs because a single intensity stimulates a point spread function (PSF) in the eye and multiple PSFs can overlap to create a single stimulus area, if the PSFs lie within the critical diameter, known as a receptive field [53]. This property is exploited by display manufacturers, stacking RGB pixels individually to produce a color pixel and in imaging methods such as half-toning. We also use this property to increase intensity resolution by sacrificing spatial resolution. Temporally, the visual system can resolve 10 to 12 images per second, perceiving them as individual images [92]. Faster moving images produce the illusion of motion in the observer's eye due to averaging of intensities over time. The critical flip frequency (CFF) is defined as the frequency at which an intermittent intensity ceases to flicker and appears as a continuous intensity. The Tablot Plateau law states that above CFF, the subjective averaging of intermittent intensities will produce the same luminosity in the visual system as an objective intensity of the same color

and brightness [119]. This is exploited by display techniques such as DLP which produce individual intensities by flipping micro-mirrors at extremely high speed to produce 8-bit intensity resolution. We also exploit this property to temporally mix intensity values to gain intensity resolution. It is also well established that spatial and temporal perception is non-separable in the human visual system [26]. A purely temporal or spatial integration of pixels will thus not suffice. We incorporate a spatial component to the temporal mixing and vice-versa to satisfy this condition. Perceivable intensity values for a given luminance is represented by just noticeable difference (JND) steps in the human visual system. On a display with $1000:1$ luminance there are about 1000 JND steps [105]. A 10-bit image should thus be visibly distinguishable on such a display to a human observer.

The CRT model: Many models have been proposed for explaining the behavior of CRT displays, based on the observable values, $F(d)$, and the predicted values $T(d)$ produced by the input digital frame buffer, d. The difference between these two curves evaluates the quality of the model. A well accepted model for CRT behavior is the gamma correction model given as [25]

$$T(d) = \left[k_g \left(\frac{d}{2^N - 1} \right) + k_o \right]^{\gamma}, \tag{4.1}$$

where N is the number of bits per channel. k_g and k_o are the gain and offset values along with γ as the tunable parameters. Though the model is accepted across all intensities, it may not produce correct results at very low intensities [25]. Our mixing methods should also produce intensities that follow

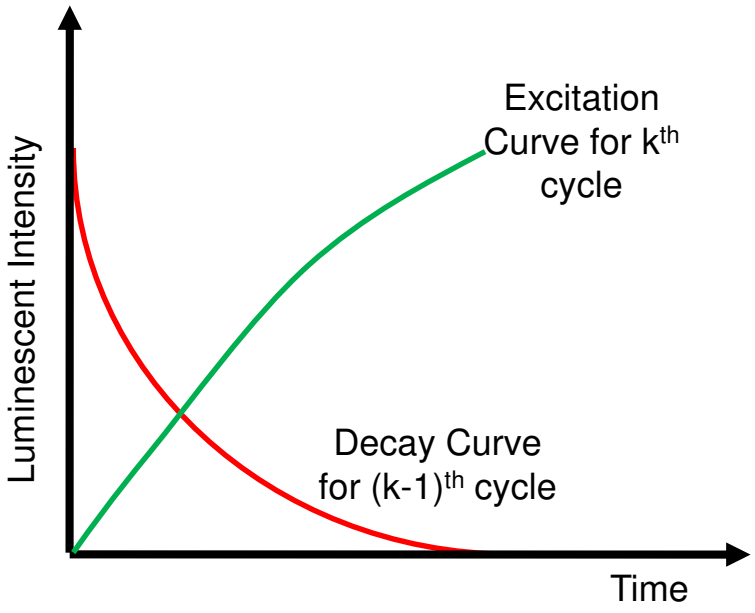


Figure 4.2 Phosphors excitation and decay curves.

this model for them to produce correct expansion of the color resolution for in-between-intensities. The behavior for decay and excitation of phosphors is also of interest, as it can affect the perceived averaging of pixel values. Phosphors response to excitation and decay is shown in Figure 4.4 for a single display cycle [87]. Adding the two curves for k and $(k-1)$ cycles gives the intensity during the excitation of the k^{th} cycle. Clearly the transition holds an average of $(k-1)$ and k^{th} intensities during excitation, this helps our mixing techniques to sustain a perceived average intensity across multiple display cycles.

4.5 Intensity Mixing to Increase Resolution

We present our mixing techniques in this section. Our methods can be applied in three ways: temporal, spatial and spatio-temporal. Temporal and spatial methods both can add 1 or 2 bits to the displayed image due to the limited frequency and resolution of current displays. Spatio-temporal method is a combination of both approaches and can provide better scaling.

4.5.1 Temporal Mixing

We sacrifice vertical refresh rate to gain intensity resolution in this approach. Given a display with a vertical refresh rate **r**, we can divide **r** into a number of sub-images such that the average of these sub-images, when shown one after the other, produce an intensity not available in the actual display. To

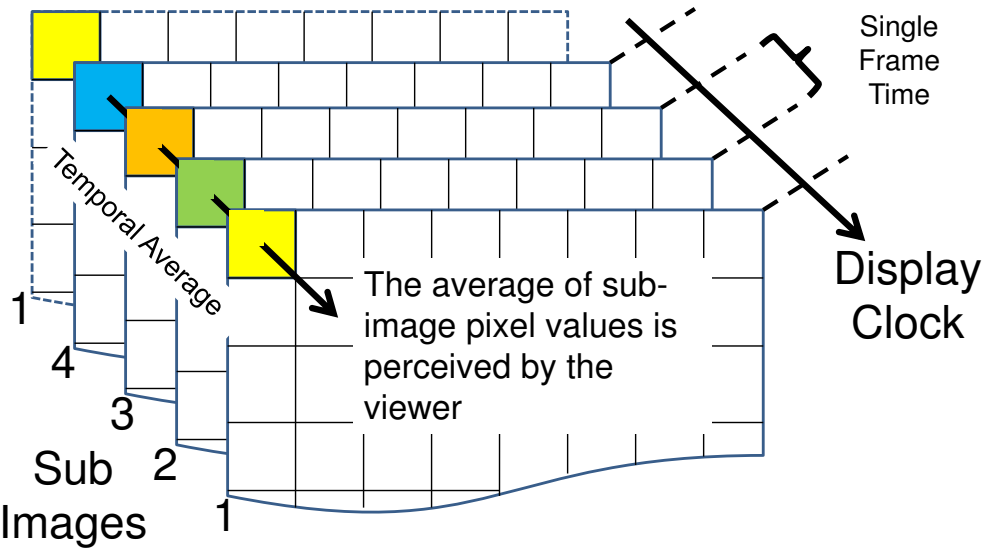


Figure 4.3 The temporal mixing, averaging over four sub-images, with each sub image being flipped at the vertical refresh of the display.

display a 0.5 intensity, for example, on a black and white display, the pixel is decomposed into black and

white pixels. These are then flipped at high speeds to produce a grey value not present in the black and white display. The intensity is averaged out temporally as described by the Tablot Plateau law [119]. From the half-toning literature, such a mixing corresponds to an amplitude modulation, albeit in the temporal domain as opposed to the spatial mixing used in half-toning [31]. The flipping speed is critical and must not be lower than the acceptable frequency perceived by the human visual system; we use a lower limit of 30 Hz. This limits this scheme to a maximum of **r**/30 sub-images. At 120 Hz such a system can show four sub-images, adding two more bits to the already existing display. The flipping must also be synchronized to the display vertical refresh such that the sub-image is changed exactly with change in the display frame, as shown in Figure 4.3.

4.5.1.1 Intensity Decomposition

Decomposition of a high-bit pixel into lower-bit pixels is needed in order to produce the desired intensity by averaging the sub-pixels either spatially or temporally. We produce a mapping for each high bit intensity such that it maps to four lower bit intensity values. More sub-images can also be used if the display refresh is high. For a typical display with 120 Hz, displaying more than four frames violates the lowest allowable limit of 30 Hz. The mapping needs to be computed once and can be an off-line process. We decompose by mapping the minimum and maximum values of the input image scale (10-bit) to the minimum and maximum of the display scale (8-bit). The intensities in the high-bit scale can then map to a sum of lower bit intensities by incrementally increasing one intensity per sub-image as shown in Table 4.1. Intensities can be assigned in multiple ways, for example for intensity 0.5, the sequence [0 0 1 1] or [0 1 0 1] or any rotations of these may be used. For a display with high refresh rate we see no noticeable difference between these two permutations. However, for lower refresh [0 1 0 1] will produce better results as the intensities are more diffused. The sub-images may be computed off-line if the input is an image but must be computed in real-time for videos. A GPU can be used for this as the process is embarrassingly parallel.

10-Bit	8-Bit Sub-Images				8-Bit
Input Intensity	F1	F2	F3	F4	Intensity
0	0	0	0	0	0
1	0	0	0	1	0.25
2	0	0	1	1	0.5
...		.....			....
5	1	1	1	2	1.25
...		.....			....
759	189	190	190	190	189.75
...		.....			....

Table 4.1 Intensity mapping for a 10-bit pixel to four 8-bit pixels

4.5.1.2 Spatial Component to Temporal Mixing

Multiple low-bit intensity combinations can be used to represent a single high bit intensity value as discussed. The redundancy produces better visual cues than a single combination shown temporally for the same intensity. Using a single mapping for an input high-bit pixel does not produce the best image for large regions of same intensity. Consider, for example, the entire screen filled with a single intensity value. Then if all pixels are allotted the same sub-image sequence, the entire screen will change simultaneously. This results in a noticeable flipping artifact and is particularly visible when the difference between the lowest and highest sub-image intensity results in a large number of JNDs. To avoid this we change the sub-image sequence given to a pixel based on its neighboring pixels. A pixel is allotted a rotation of the sequence that is different from the ones allotted to its direct north, north-west and west neighbors as shown in Algorithm 7, resulting in a spatial averaging of intensities and reducing the JND due to spatial mixing. This kind of spatial averaging relates to half-toned frequency modulation, since the pattern is dispersed over the entire region to create a better visual experience [31].

Algorithm 7 Spatial Component to Temporal Mixing

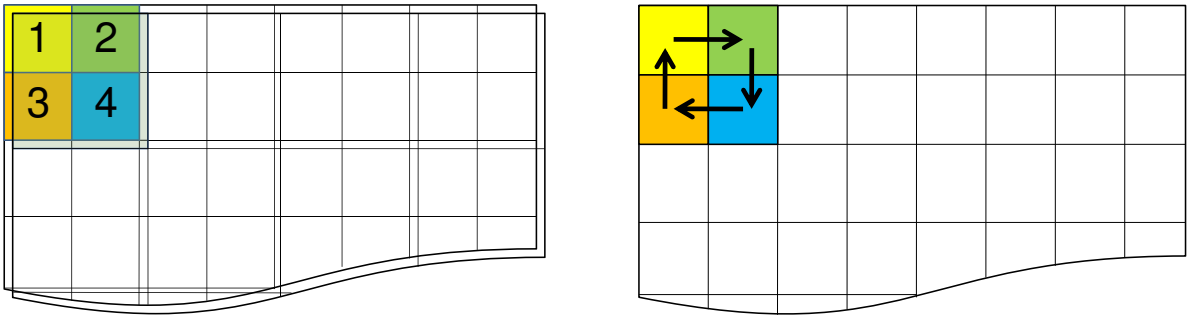
```
Allot sub-image sequence to 1^st pixel
for 2^nd to total number of pixels do
    if pixel intensity is same as North, North-West Or West neighbor intensity then
        Rotate sub-image sequence by 1 until sequence for this pixel is different from North, North-West and West.
    end if
    Save sub-image sequence allotted to this pixel
end for
```

4.5.2 Spatial Mixing

Our spatial mixing technique borrows from half-toning to amplitude modulate intensities over a fixed spatial region [31]. The resolution of the input image is increased to map each pixel to a number of pixels on the screen. Each input pixel (super-pixel) may map to a window of 2×2, 3×3 etc., such that the eye may not perceive an individual sub-pixel, but treat the group as a single pixel with average of all sub-pixel intensities, as shown in Figure 4.4(a). When viewed from a distance these sub-pixels average into a single new intensity not available on the base display.

4.5.2.1 Intensity decomposition

In our implementation we allot four sub-pixels to a single pixel of the input image. A decomposition similar to decomposition given in Table 4.1 but applied spatially can be used for this. The mapping of the input intensity to low-bit intensities remains the same. Instead of arranging into sub-images, the intensities are now arranged as a group of four pixels, with each sub-pixel allotted a low-bit intensity.



(a) Each high bit pixel maps to four low bit pixels

(b) Intensities are rotated by 1 temporally

Figure 4.4 Spatial mixing, each high bit pixel is mapped to 4 sub-pixels that are rotated by one intensity each frame of the display.

4.5.2.2 Temporal Component to Spatial Mixing

Static spatial arrangement of pixels may not provide satisfactory visual cues to perceive an average intensity, even when the sequence allotted to each super-pixel differs based on its neighboring super-pixels. This can be attributed to the fact that spatial and temporal perception in HVS cannot be decoupled. Temporal mixing of the spatial data helps to better perceive the average intensity per super-pixel and also to differentiate its identity from the neighboring super-pixels. This can be done by rotating the sub-pixel intensities by one for each display clock cycle as shown in Algorithm 8 and Figure 4.4(b). This results in better intensity perception per super-pixel and also defines the boundary between super-pixels in the input image. Current generation displays allow high spatial resolution, with up to 3M pixels. These can easily handle sufficiently large input images. However, due to the temporal dependence a high vertical refresh is also needed.

Algorithm 8 Temporal Component to Spatial Mixing

1: Allot sub-pixel sequence to 1^{st} super-pixel/
2: Allot sub-pixel sequences to rest of the super-pixels based on their neighbors according to Algorithm 7.
3: **for** All super-pixels **do**
4: Rotate sub-pixel sequence by 1 for each vertical refresh.
5: **end for**

4.5.3 Spatio-Temporal Mixing

The above methods map an input high-bit image to lower bit displays and cannot scale beyond two bits more than the base display intensity resolution due to limitations both in vertical refresh and spatial resolution achievable by current displays. They can be combined into a single system which can scale

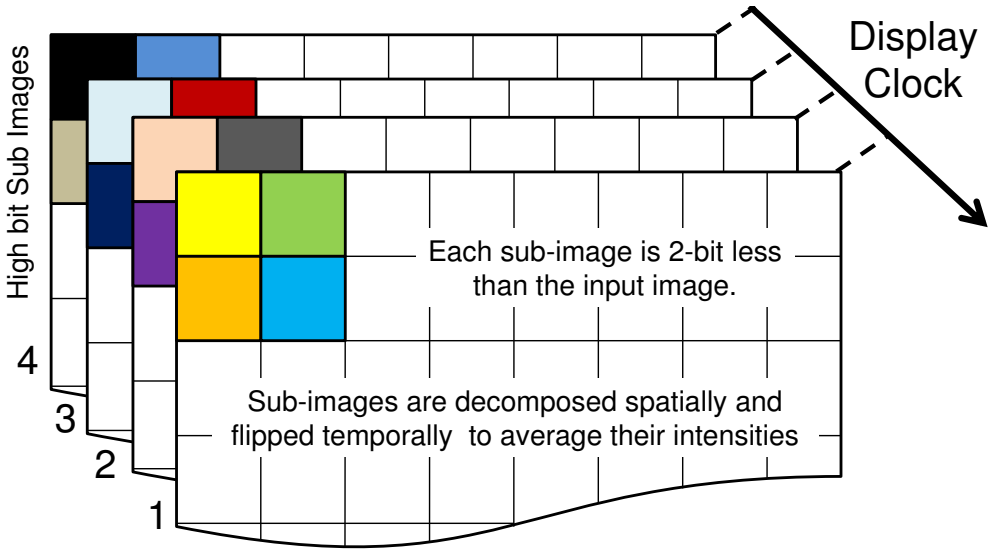


Figure 4.5 The spatio-temporal mixing, temporally averaging over four high-bit sub-images, with each sub-image decomposed spatially.

to a greater intensity resolution. This can be done by decomposing the input intensity into spatial and temporal components. For better performance, there exists a trade-off while decomposing. More bits can be decomposed using temporal mixing if the display refresh rate is high, or alternatively more weightage can be given to spatial mixing if the display provides high resolution.

4.5.3.1 Intensity Decomposition

Spatial and temporal integration of sub-pixels can be combined to gain higher than 2-bit intensity resolution by decomposing the input image into a number of sub-images. Based on the weightage given to temporal method each of these will have lower intensity resolution than the input image. These sub-images can then be spatially decomposed individually to the even lower display resolution. The sub-images when shown temporally produce an integration of spatial intensity over time, as shown in Figure 4.5.

For our implementation, we decompose a high bit pixel into four sub-images, with each sub-image pixel further decomposed into four sub-pixels, corresponding to the display intensity resolution. This can result in a maximum of 4-bit higher intensity resolution than on the base display. A higher bit intensity image, however, will be indistinguishable by the HVS for most current displays due to the presence of more intensities than the allowable number of JND steps for a given luminance value. The method, however, works if the target is a photo sensor. For example a high-bit sensor camera looking at such an interpretation of a high dynamic range input image to a low-bit display can distinguish these intensities. Such a system can thus find application in computer vision and related domains.

4.6 Experimental Evaluation

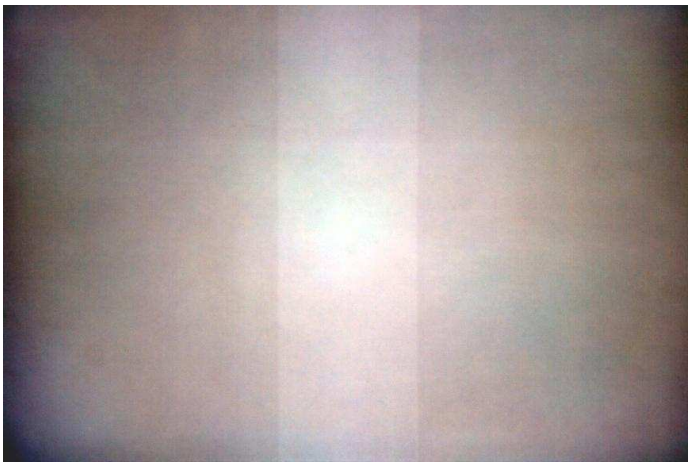
We evaluate our mixing techniques in two ways. A quantitative evaluation using a camera with a high-bit sensor is performed, the aim of which is to see the in-between-intensities and to make sure they follow the CRT gamma curve. A user evaluation is also presented in which we try to evaluate if users can perceive the generated intermediate intensities as expected based on JNDs.

4.6.1 Camera Based Evaluation

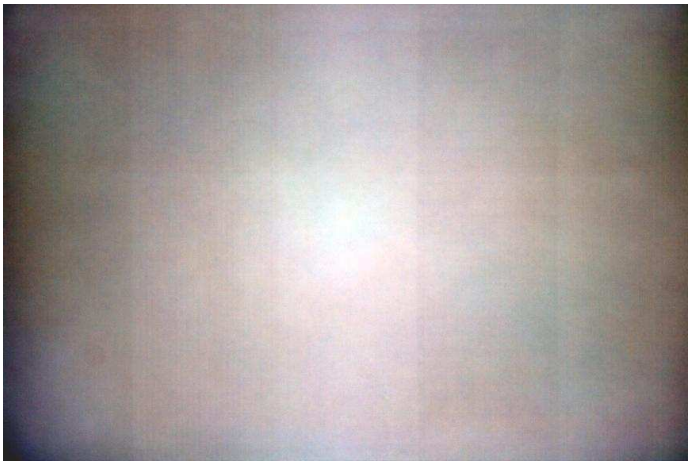
In the camera based evaluation, the intensities generated by our methods are captured using a 12-bit sensor of the Canon 350D DSLR camera. The images are taken in the RAW format to avoid color interpolation. In order to see the in-between-intensities we use a synthetic structured image displaying vertical bands of 10-bit intensities. The image is then mapped on to an 8-bit CRT display using our temporal and spatial mixing methods. The CRT is set at 640×480 pixels running at 160 Hz vertical refresh for the temporal method and 1280×960 spatial resolution running at 75 Hz for the spatial method. The spatio-temporal method uses the same display resolution and refresh as the temporal method. In order to compare perceived intensities we also display the sample 10-bit image directly on the CRT, which decimates it to an 8-bit version and makes the in-between-intensity bands indistinguishable. The image is captured using the camera which is calibrated to the intensity levels of the display by setting the lowest intensity of the CRT to zero value in the camera and the brightest CRT setting close to level 2^{12} intensity in the camera. The exposure is set to twice the vertical refresh rate of the display for the camera to see the temporal averaging and to allow the intensities to mix on the camera sensor. The input test image has 5 bands of 10-bit values. Decimation to an 8-bit scale creates exactly one intensity difference between the lowest and the highest intensity in the test image. In the 10-bit original image this corresponds to four levels of difference. The intensity bands are arranged out of order for the boundary between intensities to be prominent.

Figure 4.6 shows the input image for the 8-bit, 10-bit temporal and 10-bit spatial versions. The images shown in Figure 4.6 are contrast enhanced in order for the bands to be distinguishable in print. As seen from these images, the camera can see the 10-bit in-between-intensity levels using our temporal and spatial schemes. The levels, however, are not visible on the display using the direct 8-bit mapping. Figure 4.7 shows an MRI slice of the knee region displayed using our temporal method in comparison to a direct mapping on the base display. Clearly our method produces more detail than what is available on the base display.

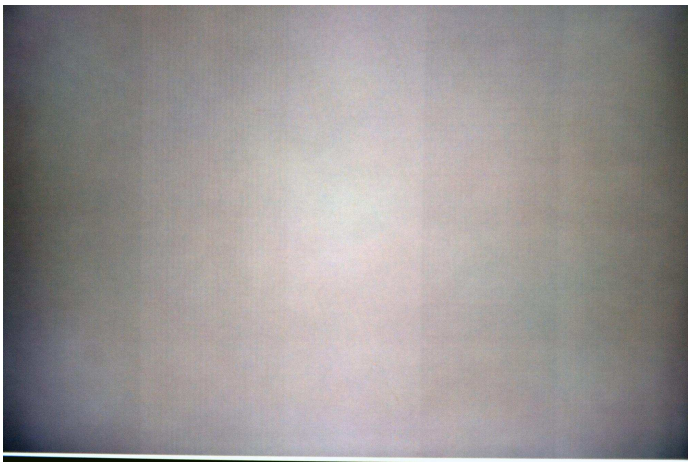
Though the intensities are distinguishable in these images using our methods, they should also follow the expected behavior of the CRT at 10-bit resolution. This is needed for our methods to successfully stretch the intensity resolution to 10-bits. In order to do this, we display a full 1024 intensity image using our methods. The intensities are incremented from left to right and from to top to bottom as shown in Figure 4.8(a). This image is displayed using our methods and is captured by the camera. Since the location for each intensity is known in the image, we can map the observed intensity, $F(d)$, to the input



(a) Displaying directly using a 8-bit display



(b) Displaying using our temporal mixing



(c) Displaying using our spatial mixing

Figure 4.6 The 10-bit input image as seen using a 12-bit camera sensor on a CRT display. The difference between the lowest and highest intensity band is 1 level on a 8-bit scale. The images are enhanced to bring out the difference in print.

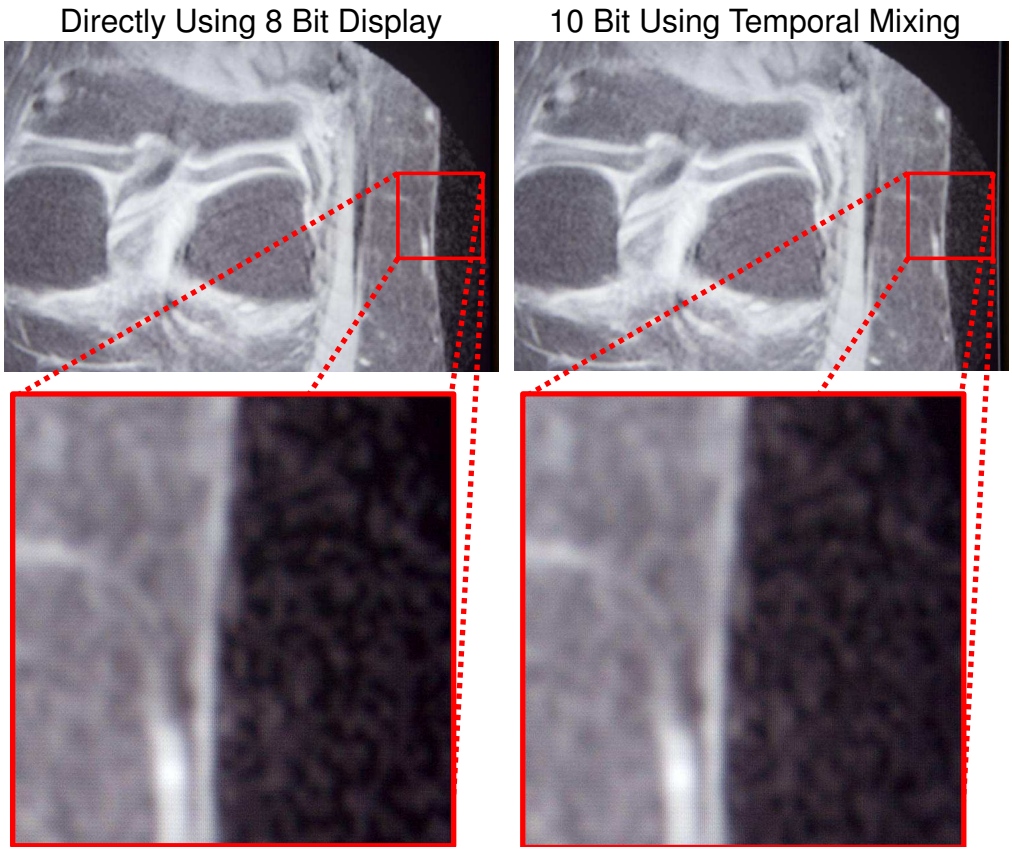
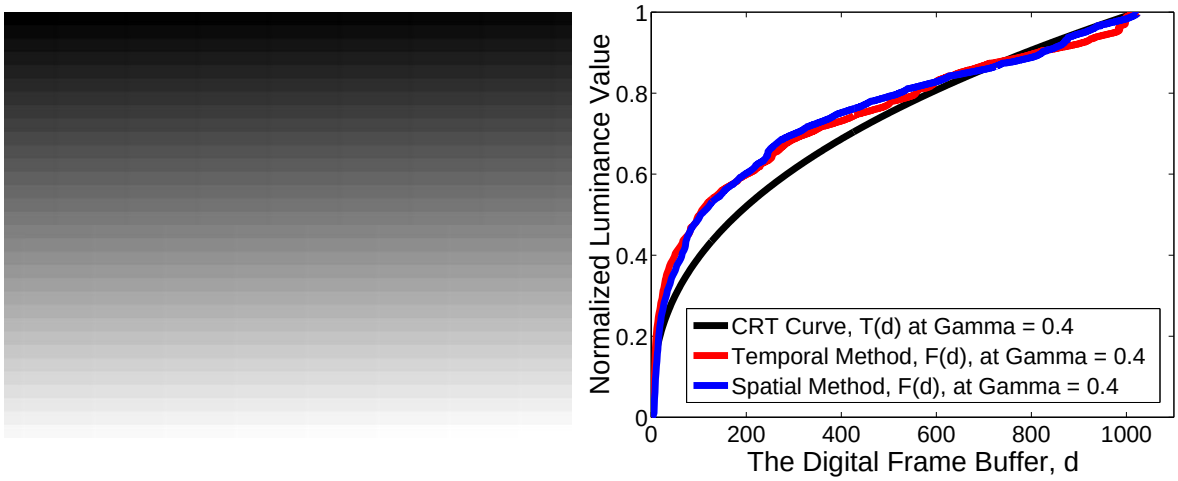


Figure 4.7 A 10-bit MRI image of the knee as viewed directly on an 8-bit display and using our Temporal method. Clearly more detail is present in the temporal method than on the base 8-bit display.

digital frame buffer value, d. This provides us with the luminance for each digital frame buffer value. $F(d)$ can then be compared to the CRT model to see if our methods produce an expected behavior.

Figure 4.8(b) shows the resulting $F(d)$ for the input d for both temporal and spatial mixing methods. The CRT expected luminance is also plotted as $T(d)$ (Equation 4.1) for a 10-bit CRT. We see that both the spatial and temporal methods follow the expected behavior projected by $T(d)$. There are, however, some variations as the observed intensities do not account for the camera model, which may alter the intensity due to aperture, exposure, optical and viewing condition dependencies. Overall, however, we see that our methods follow the CRT gamma curve suggesting that both methods produce the in-between-intensities as expected.

We also evaluate the spatio-temporal method for an 11-bit input intensity image to gain more than 2-bit intensity resolution on the display. 11-bits are used here to make sure the 12-bit camera sensor picks up the differences clearly. A 12-bit image could also have been used, however, it would be harder to distinguish the intensities using our camera sensor. The input image is again made up of 5 bands of 11-bit intensities. On the 8-bit scale, there is no difference between the lowest and the highest intensity



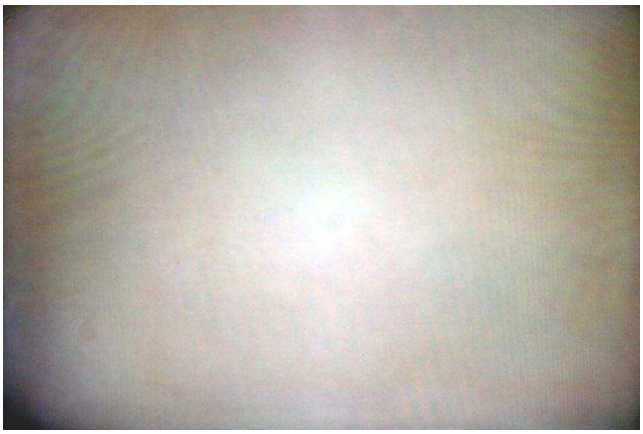
(a) The test image used to evaluate 10-bit intensity resolution

(b) The luminance curves for the 1024 level test image

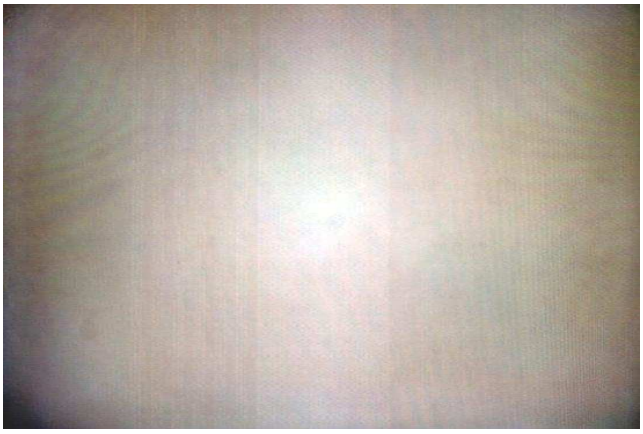
Figure 4.8 The test image used and the observed luminance values using our Spatial and Temporal mixing methods for a digital frame buffer range $0-1024$. Note the curves closely resemble the CRT gamma curve at the same gamma value. The offset of curves are due to the observing camera dependencies.

values. The 11-bit image is first decomposed into two 10-bit images, each of which is then decomposed into 8-bit images using our spatial mixing. The two images are then shown one after the other using temporal mixing. To compare our results we also show the 11-bit image directly on the CRT. The resulting images are then captured using the camera. Figure 4.9 shows the images using 8-bit direct mapping and the 11-bit spatio-temporal mapping. It can be seen that in the 8-bit mapping all intensities merge into a single value, since there is no difference between them on a 8-bit scale. The 11-bit version, however, clearly shows 5 bands of different intensity values, validating our spatio-temporal mixing. It should be stated that though the bands are visible after enhancement of the camera sensor image, they were very hard to distinguish by the naked eye, as the intensity difference is less than the consecutive JND steps.

To see the behavior of our spatio-temporal mixing on an 11-bit image, we compare the observed intensities, $F(d)$ to the CRT gamma curve, $T(d)$, at 11-bits. A sample image similar to the one shown in Figure 4.8(a) but with 2048 intensity values is displayed using our method and is captured by the camera. We plot the digital frame buffer vs the luminance in Figure 4.10. The method clearly follows the overall shape of the gamma curve, however, even with the unknown camera dependencies, it should be explained why $F(d)$ slightly diverges from $T(d)$. This can be explained by the mixing method: mixing is taking place over eight intensity values in the spatio-temporal system, each of which is gamma related to the previous intensity, thus the difference between intensities becomes less substantial than on a true 11-bit CRT, resulting in the observed behavior. It should also be noted that this may change on a different display model such as an LCD.



(a) Displaying directly using a 8-bit display



(b) Displaying using our spatio-temporal method

Figure 4.9 The 11-bit band image as seen using a 12-bit camera sensor using spatio-temporal method. The difference between the lowest and highest intensity band is not visible in the 8-bit scale. The images are enhanced to bring out the difference in print.

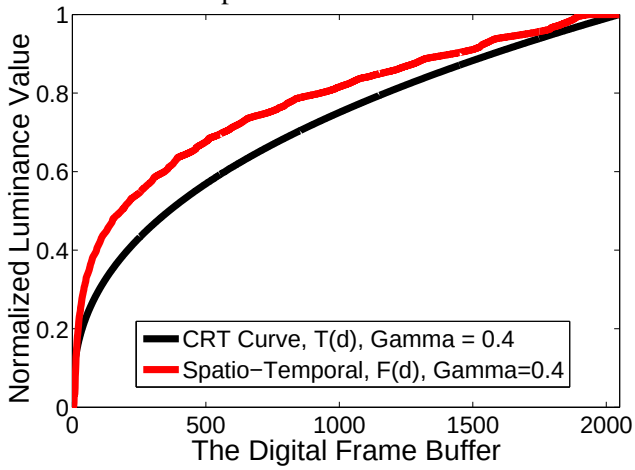


Figure 4.10 Spatio-Temporal method for a digital frame buffer range $0 - 2048$. The curve closely resembles the CRT gamma curve at the same gamma value.

4.6.2 Human Evaluation

Our experiments for human evaluation are limited to spatial and temporal methods only, as more than 1024 levels cannot be distinguished by a human on current displays due to limited number of JNDs for the given luminance. The purpose of this evaluation is to establish that in-between-intensities can be perceived by a human observer. In order to do this, we show 10-bit images with varying number of bands $(5-7)$ with $1-2$ levels of difference between the lowest and the highest intensity bands on a 8-bit scale. The test consists of 12 images simultaneously shown spatially and temporally on two adjacent CRTs. The images are randomly shown using direct 8-bit mapping to none, either or both the displays. Table 4.2 defines the experimental input used for human evaluation. The user is asked to report the number of bands visible in both displays. The images span the entire intensity range and random RGB combinations.

Task No.	Intensity Range	No. Of Bands	Color Channel	Display Bits	
				Temporal	Spatial
1	508-512	5	RGB	10	8
2	340-345	6	G	8	10
3	676-682	7	RGB	8	10
4	340-344	5	GB	8	8
5	844-849	6	RGB	10	8
6	508-513	6	RG	8	10
7	676-680	5	RGB	8	8
8	340-346	7	RGB	10	8
9	844-848	5	RGB	8	10
10	676-681	6	RB	10	10
11	844-850	7	RGB	10	8
12	508-514	7	G	10	8

Table 4.2 Experimental images used for human based evaluation.

Thirty test subjects were tested for 12 images each, the results for 8 and 10 bit spatial and temporal methods are summarized in Figure 4.11. The mean and maximum deviation from actual number of bands in the 8-bit and 10-bit images are reported for each task, with **T** specifying temporal method and **S** the spatial method. There are more errors in distinguishing 10-bit images, however, errors were also seen in the 8-bit versions. The mean error is a good indicator for the human perception of 10-bit images. We see that the mean error never exceeds two bands with the exception of task number 10. This was a particularly hard task with only red and blue channels making up the image. We found the distinction of bands in red and blue to be the hardest due to fewer transmitters and receptors for these channels in the CRT display and the human eye respectively. Green channel images were the easiest to distinguish, as seen in task T12. Over the entire experiment we found that spatial mixing produced slightly more errors than the temporal method. This may be due to the high resolution and low refresh rate used in this method. For the 10-bit images, we see a mean error of 0.8 bands with standard deviation of 0.76

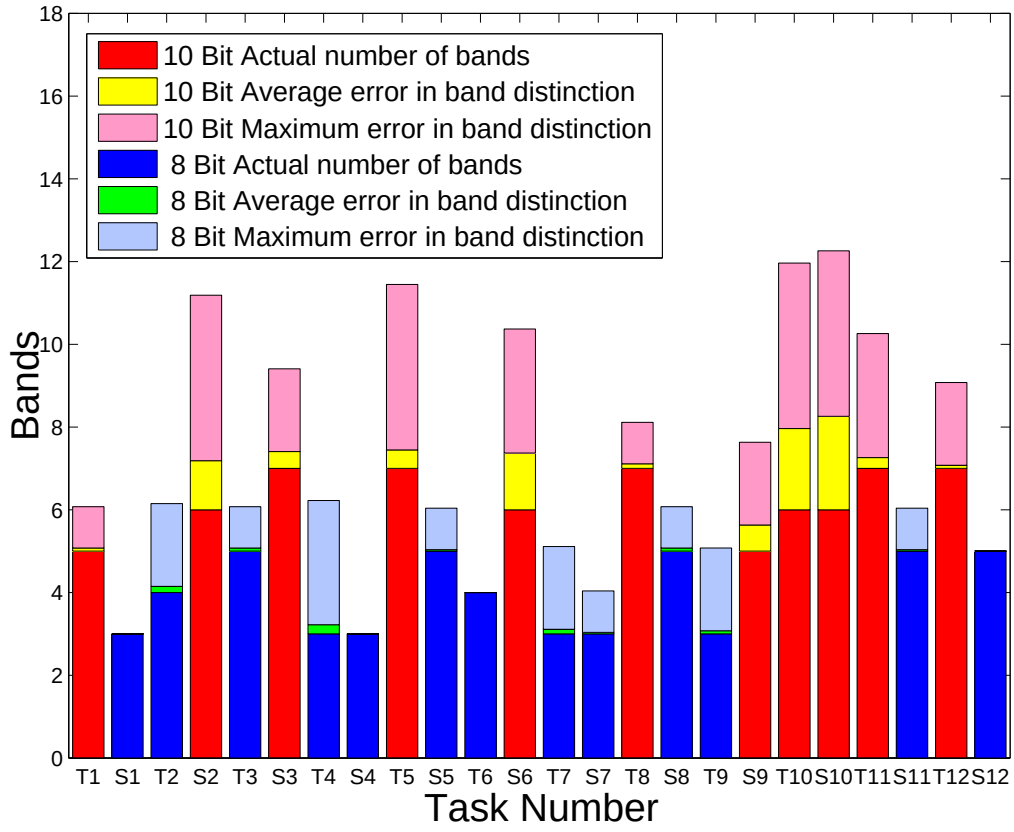


Figure 4.11 Results of the human experiment. The left column for each task represents the temporal (T) mixing while right represents spatial (S) mixing. The graph shows actual bands in the image, the average error per task and the maximum error per task. For example, for task no S6, it shows it is a 10-bit task with 6 bands, average error of 1.4 over all subjects and the maximum reported error of 3 bands.

over all subjects. Overall, our methods produced a low error for both spatial and temporal approaches and were capable of displaying more intensities to a human observer than those available on the base display.

4.6.3 Limitations of Our Methods

The main limitation of our methods for HDR displays is that they do not increase the upper intensity limit. Only the display intensity resolution is increased keeping the same brightness range. This can result in our method producing more intensity levels than the allowable JND steps. Since the mixing occurs in the eye and not on the display, an increase of more than two bits of intensity cannot be verified by HVS. Our methods are also limited by the vertical refresh and the spatial resolution of the target

display. Sacrificing the vertical refresh can produce stuttering artifacts. Lowering the spatial resolution can also cause loss in image detail.

4.7 Conclusions and Future Work

In this chapter, we presented three methods to map HDR images to a display of low-bit resolution. Our methods produce in-between-intensities that are capable of visual distinction as shown in our experiments both by human observers and by a camera sensor. Our methods can also be used for HDR videos if the video is of less than 30 frames per second. We have experimented with these by decomposing each frame into sub-images on the fly using GPUs. Our methods can visually enhance images as required by many visualization applications. The methods are technology independent and can be applied even on a high-bit per channel display to increase intensity resolution further. We would like to explore our methods for regular images and compare them with other mapping methods such as tone mapping to see if they produce better visual image quality. Advent in display contrast, resolution and vertical refresh has an increasing trend. Our methods when combined with such high contrast, resolution and vertical refresh displays can truly reproduce HDR images at enhanced intensity resolutions.

Chapter 5

The Garuda Display Wall

5.1 Introduction

The size of a display in terms of the dimensions of the surface and the number of pixels are important for visualization applications. Public displays need large area for wider viewing but do not require high resolutions. A large screen lit by a projector serves such applications. Many visualization applications, however, require large display areas and high display resolutions simultaneously to provide both detail and context. Virtual Reality environments, scientific visualization, etc., are examples where large display size with high resolutions can be useful. Such displays can be built using specialized hardware or using a cluster of computers.

Hardware to drive large displays and display mechanisms like CRT and LCD are severely limited in the maximum achievable resolution. Costs increase prohibitively beyond a few mega pixels. Tiling multiple displays is an effective way to create arbitrarily large displays for visualization applications. General purpose systems with off-the-shelf graphics accelerators can be used in a cluster to provide a cost-effective and scalable alternative for setting up large tiled displays.

There is a trade off between resolution and display size in computer displays. The resolution of the display affects the visible detail and the size affects the visual context. Zooming in to view the fine details results in a loss of the bigger picture. Zooming out bring the big picture at the expense of the details. Baudisch et al. attempt a creative solution by embedding the high-resolution portions on a screen while displaying low resolutions for the rest in an effort to achieve focus and context simultaneously [19]. Wang et al. proposed a focus-plus-context framework to magnify the features of interest [121]. These methods, however, assume that the viewer concentrates on a small region of the screen; the focus doesn't extend to the entire screen. It is observed that in a large display environment people tend to move about and change viewpoints, and therefore might be distracting, especially at the edges of the resolution-change [30]. Displays with large size and high resolution solve the trade-off between focus and context. A few efforts are underway to build very large displays. Displays with $100 - 200$ mega pixels have been built by multiple teams [3, 109, 39, 4]. These are mostly used to display static or slow-changing

images. Driving such displays directly from a graphics program will facilitate their use in interactive visualization and virtual reality applications.

In this work, we present the design of Garuda, a cluster-based tiled display wall for interactive graphics applications. Garuda system uses commodity PC hardware on Ethernet for driving individual tiles with a high-end server to coordinate them. We currently use low-end PCs on 100Mbps network to drive the tiles. Garuda's software system is built on the Open Scene Graph API [9]. Any application built on OSG can be rendered transparently on the display wall without any modification. Garuda caches and manages the transmitted geometry at the rendering clients to exploit temporal and spatial coherence in the scene. This sets it apart from solutions like Chromium [55] whose demands on the network are very high. Wallace et al. describe the design issues behind the Princeton display wall from the point of view of scalability to videos, images, graphics and audio [120]. A recent survey on large, high-resolution displays cite scalability to large tiles, high-performance rendering, and integration into a computing environment as among the top ten research challenges faced today [82]. The design of Garuda has addressed these issues directly.

The main contributions of the Garuda system for tiled-displays are two: (a) applicability to a large scenarios and (b) scalability of the hardware and architecture to very large displays. For wide applicability, we provide the ability to render any Open Scene Graph application to the display wall without modifications. Scalability in hardware is achieved by the use of low-end PCs on commodity Ethernet as the rendering nodes, keeping costs low. Scalability in the architecture has multiple components. An adaptive algorithm at the server culls the object hierarchy to the frustum hierarchy optimally to determine the objects inside each tile's frustum. The caching of the partial scenegraph at each node facilitates the transmission of only the new objects in each frame, keeping the network requirements low. As the number of tiles increases, caching becomes more effective as fewer objects are visible to each tile. Lastly, the objects are sent to the rendering nodes using a multicast protocol to further reduce the network load. Since the server is on one end of all network traffic, multicasting can send an object to all clients in about the same time needed to send to one client. As the number of tiles increases, more objects will be visible to multiple tiles, bringing greater gains using a multicast protocol. Garuda system exploits the power of distributed rendering. No node in the system including the server needs to render the entire scene. This facilitates the rendering of very large scenes in a distributed manner. The server deals only with the bounding volumes and each client only renders the sub-frustum corresponding to its tile. As a result, a 2×2 system could render the full Powerplant model at about 50 fps and above at a resolution of 2048×1536. A single node of the same capacity achieved only 10-12 fps rendering a 1024×768 image. A 4×4 system using low-end machines with ATI Xpress 200 motherboard graphics achieved 10 fps on the same model, with a display resolution of 4096×3072.

This chapter is organized as follows. The related literature is reviewed in Section 5.2. The design of Garuda system is given in Section 5.4. Transparent rendering for OSG applications appears in Section 5.5 and experimental results from it in Section 5.6. Discussions and conclusions are presented in Section 5.7.

5.2 Related Work

We present a review of the literature related to the construction of large displays in this section. See the recent survey on the hardware, software and human-factors aspects of large, high-resolution displays [82] to get a comprehensive review of the options for building such displays.

Large graphics display systems have been built using specialized hardware by companies like Silicon Graphics. Specialized implementations such as Power Walls [10] and CAVE [36] exist at several places. Sepia by Heirich et al. use specialized hardware for image combining over a high speed, high bandwidth network [76]. Metabuffer by Bajaj et al. is a hybrid software cum hardware approach for color combining and blending to a tiled display [125]. Though these systems achieve interactive framrates, they are expensive, have poor scalability to large sizes, and require expert-level maintenance and setup. Cluster-based solutions for creating large displays have gained a lot of interest recently [32, 55] because of their low reliance on specialized hardware and low mentainace cost. These systems consist of a number of commodity PCs that are interconnected over a LAN or via low-latency networks like the Myrinet [10, 55, 11]. Such prototypes have shown to address various issues such as seamless tiling and color-balance [30, 74]. Cluster-based display can also scale resolution using two approaches: *master-slave* and *client-server* [32].

In the master-slave setup, the dataset is mirrored across all the nodes and multiple instances of a program run in parallel, one on each node. Their running is synchronized such that they assume identical behaviors at synchronization boundaries. Each node renders the entire scene but displays only a certain portion of it. Bressoud et al. proposed Hypervisor, a system-level program synchronized approach, that treats an actual software system as running on a virtual machine close to the actual microprocessor architecture [28]. VR Juggler, a framework for virtual reality applications, follows an alternate application-level program synchronized approach in which the responsibility of synchronizing lies with the application [23]. Net Juggler is an open source library that turns a commodity cluster running the VR Juggler into a single image cluster [15]. The master-slave approach assumes that each node in the cluster would be able to render the entire environment in its entirety. This runs counter to the motivation of load-balancing that is critical to cluster-based displays. It is also difficult to handle dynamic environments since the data is replicated.

The client-server models store the dataset on a central server. The server can also use a distributed data management framework, as in [45]. The server distributes appropriate data to each client node and performs the synchronization among the rendering nodes. The data distribution can follow sort-first, sort-last or a hybrid k-ary distribution strategy [100, 99]. Data can also be intercepted at the API level [55] or the display manager level [2]. Hyperwall [101], VisWall [11], LionEyes Display Wall [7] use the Chromium model [55]. The plus point of Chromium is that it can clusterize any application transparently, however, fails to capture the coherence across frames. This leads to high network requirements even when the scene remains unchanged.

Another approach to the client-server architecture is to carry out the distribution at the 3D object level. Virtual Graphics Platform (VGP)can handle 3D objects transparently for OpenGL applications.

ModViz Renderizer software, based on VGP, can render any OpenGL Performer application to a tiled display with some code modification [8]. The approach followed by Syzygy [103] and OpenSG [118] for display wall rendering exploits scene structure along with 3D objects for even better performance. Germans et al. presented Aura which has both options for master-slave and client-server approach[46, 116]. Corrêa et al. described iWalk, an out-of-core rendering scheme using visibility-based prefetching of data in massive models [34]. Blue-C Distributed scenegraph by Naef et al. uses a scenegraph node distribution approach with some user interaction [78]. The Garuda system follows a server-push philosophy which allows us to exploit temporal and spatial coherence during computations. The server-push philosophy also helps in handling dynamic objects in a scene. Garuda distributes the scenegraph nodes across multiple clients automatically in a manner such that all nodes common to all the clients are multicast only once and existing nodes at clients are not sent again, keeping the network load minimal.

5.3 As a Computational Display

The design of Garuda utilizes distributed rendering to render to individual tiles. Each client is Garuda consists of a PC connected to a conventional monitor stacked into the display wall. The design thus follows the Computational Display framework given in Section 1.9 in a distributed fashion (Figure 5.1). The image is generated by the physical tiling of conventional monitors. This produces a large resolution image using computation. The algorithmic distribution is hybrid, both distributed and single channels are used in Garuda. The server is used as single computation hub, which culls the scene and sends it to clients and the clients render independently, utilizing parallel rendering capabilities controlled by the server, resulting in a distributed rendering of the scene.

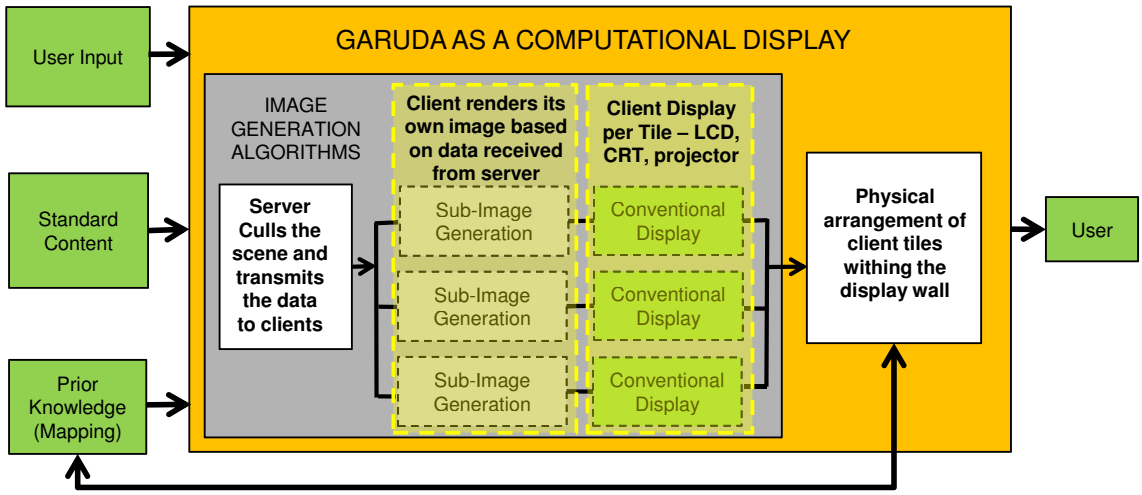


Figure 5.1 Garuda as a Computational Display.

5.3.1 Physical Processes Involved

Garuda uses a physical arrangement of tiles to produce a large scale image. The physical process involved is simple tiling of conventional displays. Each tile in Garuda corresponds to a client which is connected to the server over standard Ethernet. The location of each client within the larger display is known based in which images are rendered on each tile. This information is also used to cull the scene to individual clients to balance the load distribution.

5.3.2 Algorithmic Load Distribution

Multiple load distribution levels are employed in Garuda. The object level representation of scene (as Open Scene Graph) helps exploit the 3D structure of the scene. Culling of the scene, performed by the server, for each tile helps exploit scene coherence. Each client is assigned a cache which helps exploit temporal coherence. And finally the parallel rendering of the scene exploits distributed rendering capabilities of the system. Garuda thus follow all possible load distributions outlined by the Computational Display framework. Each of these aspects of the Garuda design are explained in greater detail in the following sections.

5.4 Garuda: A Geometry-Managed Display Wall

Two considerations guided the design of the Garuda system: scalability to large displays and usability to a number of applications. The system follows a client-server approach (see Figure 5.2), but additionally exploits the temporal and spatial coherence in the scene structure to minimize the network resource utilization. The Garuda system consists of a server that coordinates all activities and several rendering clients, currently one for each tile, that perform the rendering[1]. The server is a high-end machine while the clients are built using commodity computers. We currently use low-end PCs in our experiments; clients with better graphics and memory resources can directly enhance the performance of the overall system. No client nor the server needs to render the whole scene in our design. The client nodes and the server are connected using a standard $10/100$ Mbps Ethernet.

The user application, built on an Open Scene Graph API, runs on the server machine. The interception mechanism takes the viewpoint from the running application at every frame in response to keyboard or mouse input. The server has a reference pointer to the entire scene loaded into memory by the running application as a scenegraph and determines the objects that are visible to each tile using an adaptive view frustum culling algorithm described later. (One of the easy extensions to the system is the use of out-of-core rendering at the server to handle truly massive environments.) The objects not already present in the cache of the clients are sent to each using a multicast approach. Multicasting can

[1]The tiled-rendering literature refers to the rendering nodes as tile-servers and the node where the application runs as the tiling client [55, 116]. We use a traditional interpretation of server and client in this work and have multiple rendering clients and one tile server that coordinates them.

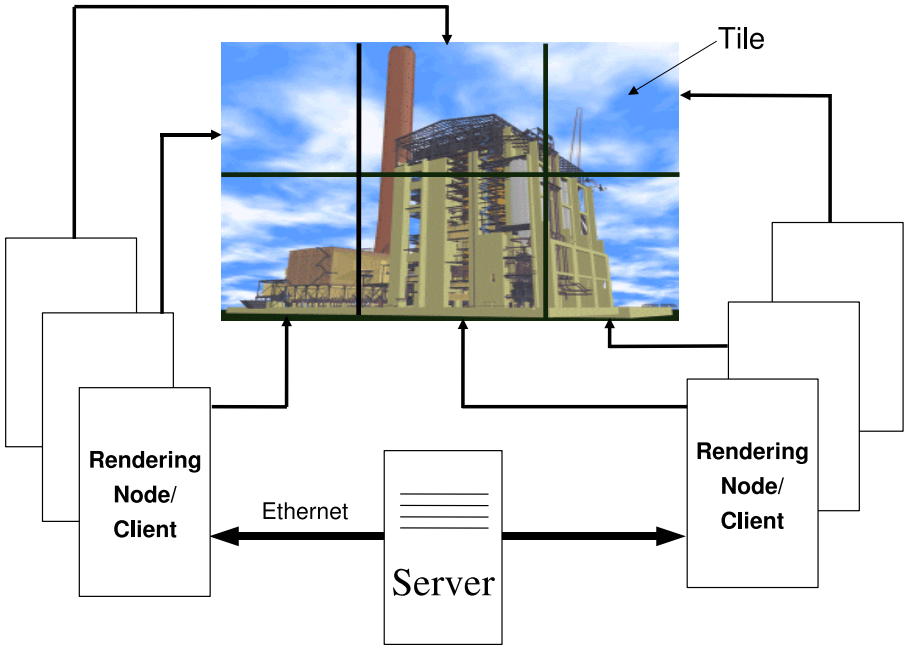


Figure 5.2 Schematic of the Garuda hardware system. The server runs the user application and controls the rendering and synchronous display of the rendering nodes of the clients, each of which draws a single tile. The server is connected to the clients over commodity Ethernet network.

exploit the spatial coherence of objects that intersect multiple tiles. The clients cache these objects so as to reuse them subsequently. This exploits the temporal coherence in the visible models in each tile. The clients start rendering a frame immediately after receiving all necessary data and inform the server when they are ready to swap the buffers. The server synchronizes the displays by ordering a common swap after receiving the ready-to-swap messages from all clients. Garuda facilitates transparent tiled display of any Open Scene Graph-based application without the user having to modify anything. This is accomplished by replacing the cull-draw-swap actions of the Open Scene Graph system using Garuda's cull, distributed draw, and synchronized swap mechanisms described later. Figure 5.3 shows the tasks undertaken at the server and client along with the flow of control between them for each frame rendered using our system. We now explain the different steps involved in Garuda's operations.

5.4.1 System Startup and Initialization

The Garuda library is loaded before running any OSG based user application at the server machine. The init function in the Garuda library initializes all the client connections and pre-processes the scene-graph before entering the rendering loop. The user application loads the scenegraph into memory whose reference pointer is stored with the server. The server computes the oriented bounding boxes (OBB) for all nodes in the scenegraph using a simple PCA approach. OBBs are used for view-frustum culling by the server at each frame in the rendering loop. A generic OSG-based viewer application is fired up at

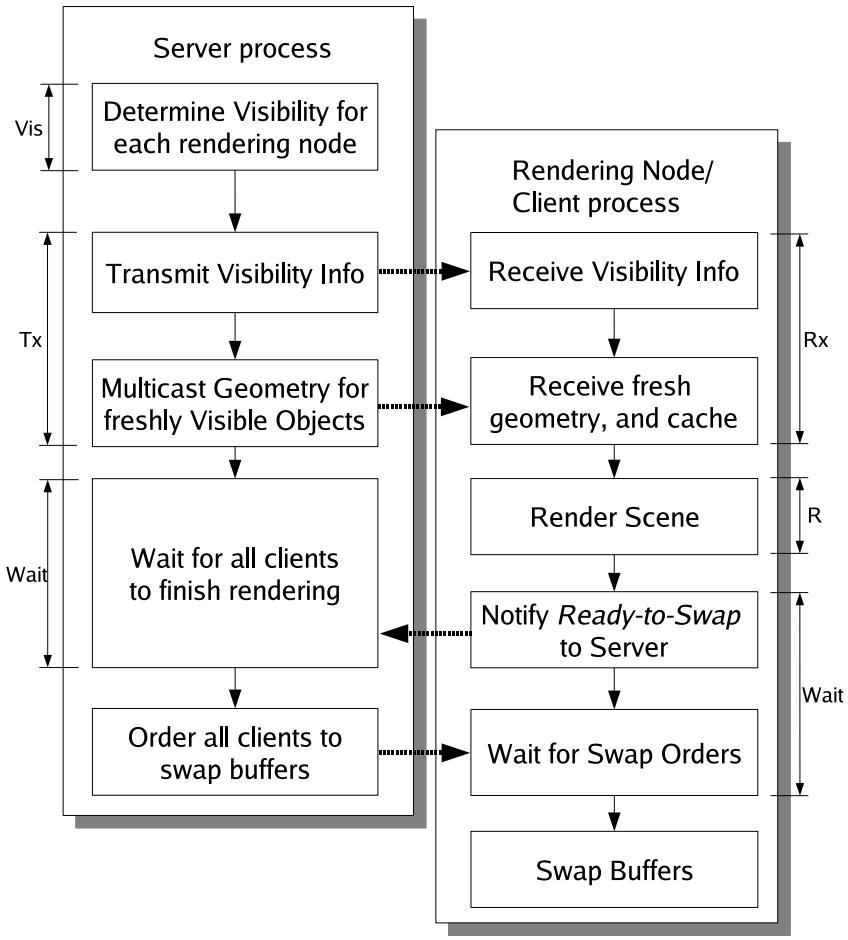


Figure 5.3 Server and client side processing and control flow for each frame. *Vis* refers to the visibility determination stage, *Tx* and *Rx* refer to the transmission stage, and *R* denotes the rendering stage. Pipelining of these stages is not shown here.

each client node. A simple client-daemon enables the server to initiate this process over the network. The client application receives scenegraph data and viewing parameters from the sever incrementally and performs the steps described below.

Data transmission starts when the first frame is rendered. The cull traversal of the first frame by OSG will result in the Garuda server determining the objects to be sent to each tile. A partial scenegraph is constructed at each client as objects are received from the server. The package of data sent to the clients contain the geometry nodes for the visible objects as well as the entire path from the scenegraph root to them. Transmission, rendering, and synchronization of the first frame is handled like all subsequent frames as explained in Section 5.4.2. Rendering the first frame involves large data transmission due to the starting up issues; subsequent frames take less time as only the incremental portions of the scenegraph needs to be sent.

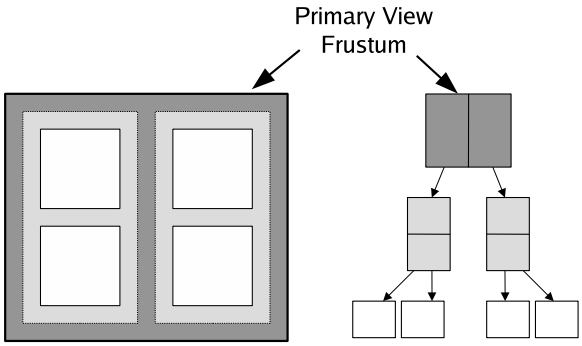


Figure 5.4 Frustum hierarchy is built by dividing the primary view frustum using horizontal and vertical planes successively. Each division creates an additional level in the hierarchy. The near and far planes are common and do not play any role.

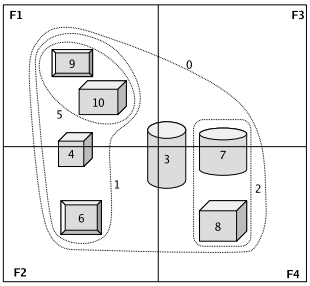


Figure 5.5 Hierarchy of objects as visible to a 2×2 tiled arrangement of view frustums. The grouping of objects is shown. F1, F2, F3 and F4 represent view frustums. Their adaptive culling is shown in Figure 5.6

5.4.2 Rendering Pipeline

The user or the application program need not be aware of the tiled display when using Garuda system as any Open Scene Graph based application can be displayed on it without any modification. The system expands the user's view volume – called the *primary view frustum* – to fill the complete tiled display, creating internal sub-frustums for each tile. The primary view frustum is divided into equal-sized tiles in accordance to the arrangement of tiles in the display wall. The user controls the viewpoint and can modify the scenegraph. The cull and draw operations of the user program initiate special processes that render the scene to the tiled display. Garuda's rendering pipeline can be divided into three stages: Visibility determination, data transmission, and rendering and synchronization. These are the operations that determine the overall performance of the system and are pipelined to maximize the rendering performance. We describe each stage now.

5.4.2.1 Visibility Determination

Visibility needs to be calculated for each tile's view frustum for tiled rendering. We use an adaptive view frustum culling algorithm to do the tile-sorting. This algorithm culls the objects of the scenegraph to each tile's sub-frustum in every frame. We do not use visibility culling as the temporal coherence is better exploited with frustum culling than true occlusion culling for a tiled display. This is because the occluded objects in the view frustum can become visible in subsequent frames. It is thus safe to send such objects to the tiles in the first place. The tile-clients can do occlusion culling to reduce rendering load. Our philosophy of using low-end clients for scalability, however, makes this difficult.

Garuda's visibility algorithm deals with two hierarchies: the Object Hierarchy (OH) of the scenegraph and the Frustum Hierarchy (FH) of the primary and sub-frustums. The Frustum Hierarchy is

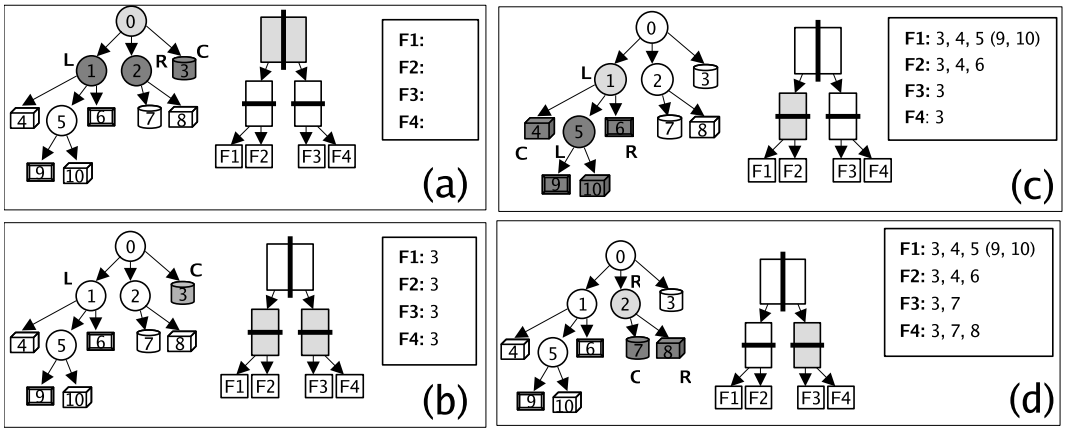


Figure 5.6 Adaptive culling of the scene structure in Figure 5.5. The object and frustum hierarchies are shown along with already determined visibility list. Working nodes are shown as light-gray. Dark gray objects are the ones that need to be recursed further. (a) OH root is classified as per the bisection plane of the FH root. L, C, R classification is shown. (b) Continuing culling for set C. (c) Continuing culling for set L. (d) Continuing culling for set R

obtained by bisecting the primary view-frustum recursively – alternately using vertical and horizontal planes – until each individual tile's view-frustum is reached (see Figure 5.4). The bisection plane used at each level is saved at the internal nodes in the FH. The algorithm traverses the OH and the FH adaptively to minimize the number of frustum-box intersection tests. The process begins by marking all objects visible to any tile using conventional culling of the scenegraph to the primary view frustum as shown in Figure 5.5. The `adaptive_OHandFH_Cull(OH_Node, FH_Node)` algorithm is invoked with the root `OH_Node` of the culled scenegraph and the root `FH_node` of the frustum hierarchy.

Algorithm 9 adaptive_OHandFH_Cull(OH_Node, FH_Node)

```
1:  if leaf(FH_Node) then
2:      Mark OH_Node as visible to FH_Node
3:      return
4:  end if
5:  [L, C, R] ⇐ ClassifyLCR(OH_Node, FH_Node.plane)
6:  for all c in set C do
7:      adaptive_OHandFH_Cull(c, FH_Node.neg)
8:      adaptive_OHandFH_Cull(c, FH_Node.pos)
9:  end for
10: for all l in set L do
11:     adaptive_OHandFH_Cull(l, FH_Node.neg)
12: end for
13: for all r in set R do
14:     adaptive_OHandFH_Cull(r, FH_Node.pos)
15: end for
```

In Algorithm 9, if `FH_Node` is a leaf, it represents a tile, and the corresponding `OH_Node` is marked to be visible to it (line 2). If `OH_Node` is a leaf, `FH_Node` needs to be unfolded to its sub-frustums. However, if neither `FH_Node` nor `OH_Node` is a leaf, a decision needs to be taken to determine which of the two hierarchies to unfold next. Line 5 uses `ClassfiyLCR`, an auxiliary function which groups the children of `OH_Node` into three sets: L (Left), C (Cuts) and R (Right), with respect to `FH_Node`'s bi-section plane as shown in Figure 5.6. Classification to these groups is done using OBB of the `OH_Node` instead of its actual geometry as it is a compact and conservative representation for it. Note that the sets L, C and R are disjoint. So, objects in set L need to be tested further with the left sub-frustum only (lines 10–12), whereas those in set R need to be tested with the right sub-frustum only (lines 13–15). Objects in set C cut the bisection-plane and hence must to be tested with both the sub-frustums. In a typical scene, a majority of objects fall in sets L and R, thereby potentially reducing the computations by half. The algorithm adapts to the scene structure and the viewpoint, following an optimal route to assign objects to a tile using the shortest path. If N is the number of objects in the primary view frustum and M the number of tiles, the algorithm has a worst-case running time of $O(MN)$ when all objects intersect all tiles and a best-case running time of $O(\log M)$ when all objects are in one tile. The average running time depends on the branching factor of the scenegraph and the number of nodes that intersects the frustum plane in each step. For good hierarchical scenes the running time varies in practice as $O(\min(N \log M, M \log N))$. The culling time for a hierarchical version of the Powerplant model for a 4×4 tiled display is under 4 milliseconds (see Section 5.6). More analysis and details on the culling algorithm including comparison with other approaches can be found in [84].

5.4.2.2 Transmission

The visibility determination stage identifies the OSG nodes present in each tile's view-frustum. Garuda server needs to send these nodes to the respective clients and coordinate their rendering for every frame. Transmission over the network is a serious limiting factor in all cluster-based display systems. The performance of this stage is determined by the network bandwidth and latency. The server updates the visibility information at each client by sending it the list of objects in its frustum for the next frame. This information in the Garuda system is less than 512 bytes per frame for each client. The visible objects that are not present already in each client's cache are subsequently send to it next. The server keeps track of the cache state of each client and has the necessary information for this. These nodes to be sent are serialized into memory buffers using OSG's file-writing mechanism; this data can optionally be compressed. The serialized data is then multicast over the network to a multicast group that includes all clients. The client nodes gather the necessary data from this multicast, deserialize the geometry, add it to the scenegraph and cache it for later reuse. The server keeps track of the objects that have been transmitted to each client to avoid retransmission in later frames. A simple handshake mechanism ensures that the data is transmitted reliably using the UDP multicast. The protocol overheads are low as packet losses are insignificant even on the UDP as the system is built as a tight LAN sub-network.

The alternative to multicasting is the use of unicast using a more reliable TCP protocol. Multicasting ensures that the sever transmits data only once even when an object is needed by multiple clients. This situation is common for tiled displays, especially as the number of tiles increase and the tile size shrink. This is a critical aspect for the scalability of the Garuda system. Multicasting thus ensures that the network requirements don't scale linearly with the number of tiles. The network requirements remain practically constant for different tile configurations. A demonstration of this for the initial startup of the display wall – when the network requirements are the highest – is given in Section 5.6. A unicast-based scheme needs to send each object separately to each client that needs it, increasing the network requirements.

All communications between the server and the clients are executed in separate threads or processes in the respective computers. This ensures that the communication doesn't hold up other activities. Integrity of the data is checked before the received data is used, to make sure all of it is available.

5.4.2.3 Rendering and Synchronization

The clients start rendering the scene for a frame when all objects for it are available. The time taken for this step is proportional to the geometry inside the view frustum. This can vary from client to client. A server-coordinated swapping of the frame buffers is, therefore, used for synchronizing the display to ensure simultaneous change. Each client sends a *ready-to-swap* signal to the server after rendering to the back buffer is completed. The server sends a *swap* message to all clients after receiving ready-to-swap from all of them. Network latency is crucial for this simple synchronization procedure. Our experiments show that a commodity Ethernet can synchronize satisfactorily at interactive frame-rates.

5.4.3 Scenegraph Management

Data sent to the client contain the position of an object with respect to the root node in the scenegraph along with its geometry. The client creates the tree structure from the data received and inserts each node at the appropriate location in its scenegraph. As more and more objects are sent to a client, its scenegraph begins to resemble the overall scenegraph present at the server.

The client also caches the geometric objects received. The cache enables our system to exploit the inter-frame coherence of data. Each client has a fixed cache and uses an LRU algorithm to remove objects from it when the cache gets full. Only the geometry nodes are cached and removed; the internal nodes of the scenegraph are retained. Eventually, the structure of the server's scenegraph will be present at every client, but with only fewer leaf nodes. Such a replication of scenegraph structure helps for consistent and incremental scenegraph management by the server.

The caching and the exploitation of temporal coherence is another important step towards scalability to a large number of tiles. As the number of tiles increases, each client needs to render less and can hold more data in its cache. The server-push philosophy reduces the client's load and hence low-end clients can be used.

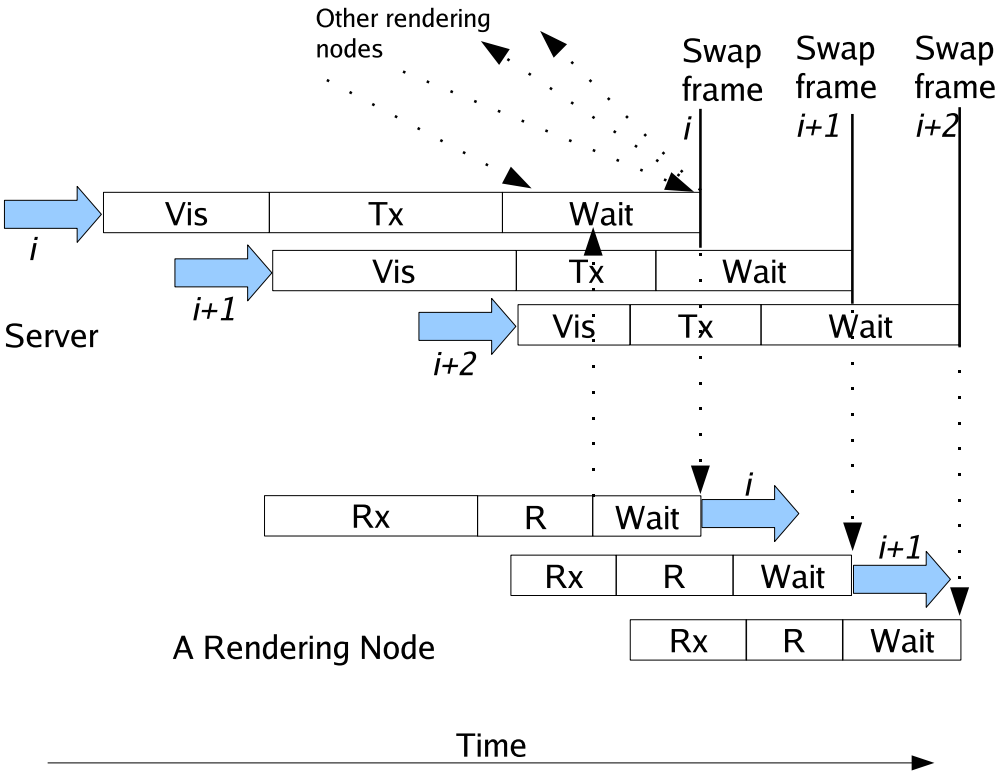


Figure 5.7 Inter-frame pipelining of the different stages. Vis denotes the visibility determination, Tx the geometry transmission, Rx geometry receiving and R the rendering. Here i denotes the current frame number. The effective FPS of the system gets increased due to this interleaving.

5.4.4 Pipelining

The three stages of Garuda's rendering pipeline (Section 5.4.2) can be pipelined. While the server is waiting to order swap for frame i, it can perform the visibility computation and transmission for the next frame. The client, on the other hand, can receive data for the next frames when frame i is being rendered. Figure 5.7 illustrates this pipelining between the various stages of the server and a client. We implement pipelining by taking advantage of the pipelined App-Cull-Draw of the Open Scene Graph system.

5.4.5 Handling Dynamic Objects

Handling dynamic objects properly is a big challenge in any master-slave framework. Yet, the possibility of some objects changing their positions or orientations is critical to many applications. The Garuda system handles this using the dynamic transformation nodes of Open Scene Graph. Selected intermediate transformation nodes are monitored every frame for any change in their parameters. If the parameters change, the composite transformation matrix at that node is extracted and sent to all clients

that have the node using a special notification mechanism. The client will replace the matrix with a new one and the object will appear transformed when the next frame is drawn. This mechanism can handle all dynamic and articulated objects which are the most common form of dynamic objects. Dynamic objects could also include those that change their shape or appearance. Since the geometry itself changes in this case, the server can order the deletion of the corresponding nodes. The new object added will automatically be identified and transmitted using the normal visibility mechanism.

5.5 Rendering Transparently to a Tiled Display

A central issue concerning a tiled display wall system is its universal applicability. One would not want to specially modify an application for rendering it to a tiled display, as it would severely restrict the applicability of such a system. The utility of a tiled display system is high only if any application developed using a standard graphics/visualization API can be rendered automatically to it. Chromium system manages to do this for any OpenGL application by intercepting all OpenGL calls and sending the primitives to the clients. However, it does not cache the data easily at the rendering nodes. This increases the network requirements heavily, especially for scenes with huge geometry.

The Garuda system is designed to automatically render any application built on the Open Scene Graph API [9], without modifying the application in any way. A scenegraph structure enables the caching of objects at the clients to exploit the inter-frame coherence of data and helps reduce the network traffic. The scenegraph API also provides the mechanism to pipeline the app-cull-draw stages to enhance the throughput of the system. A standard API like the OSG has a wide user-base which can benefit from transparent tiled rendering. Other scenegraph APIs like OpenGL Performer [93] can also be used. We selected OSG for its rising popularity and the open-source development model. Transparent, tiled rendering is provided by intercepting relevant calls of the OSG API and replacing them with our own. Since OSG maintains high level objects as nodes, the intercept mechanism is less computation-intensive and it is possible to optimize the usage of network bandwidth by caching OSG nodes at the clients.

An OSG application typically loops through three main stages: App, Cull, and Draw. For transparent rendering, the Garuda system intercepts the calls to these at the OSG API level and performs the culling, drawing and synchronization steps needed for the tiled display. Figure 5.8 shows the normal control flow for OSG and the control flow adopted by our system. Garuda's actions in each stage are described below.

1. **App:** The application stage includes keyboard/mouse event handling, scenegraph manipulations, animations, etc. This is how the user controls the viewpoint and manipulates the environment. Garuda does not alter this stage and leaves it under the user program's total control.

2. **Cull:** The cull stage performs visibility culling of the scene graph in preparation for rendering. Garuda overrides OSG's native culling to perform the adaptive OH and FH culling as discussed in Section 5.4.2.1. The results of the culling are the list of visible objects for each tile and the list of objects to be sent to each client.

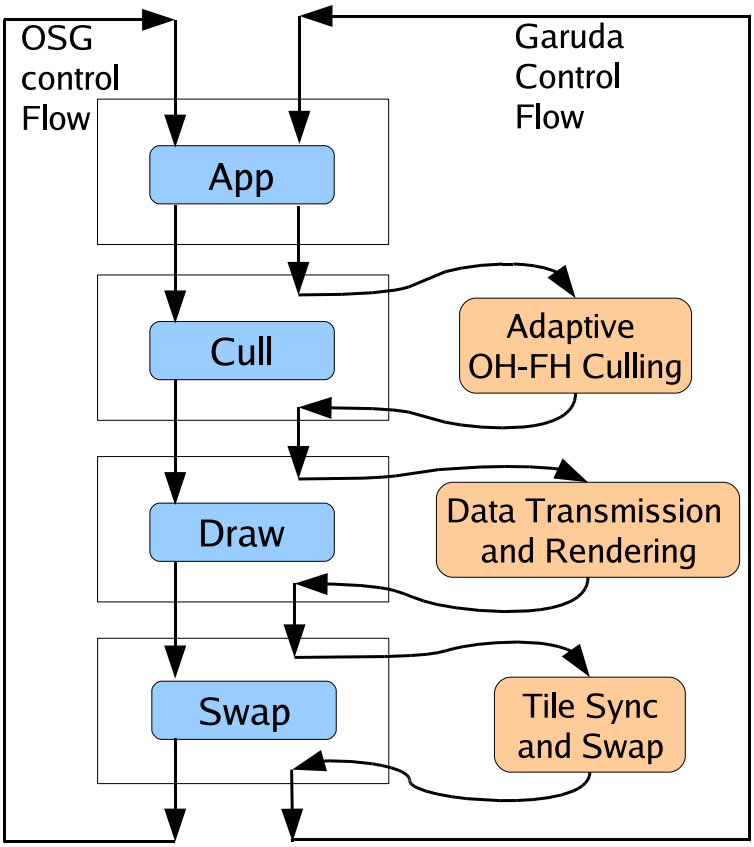


Figure 5.8 The cull, draw and swap steps of OSG's default control flow (left) are intercepted and replaced in the control flow (right) so the user program doesn't need to be modified for tiled rendering.

3. **Draw:** The draw stage renders the visible portions of a scene graph. Garuda replaces OSG drawing stage with rendering on the tiles. All scenegraph data required for rendering is transmitted in this stage (Section 5.4.2.2). The clients draw their scenes when all data is received but postpone the buffer swap until notified by the server.

These stages are followed by a call to swap the display buffers, available with the windowing API. Garuda intercepts the swap call at the server (which is strictly not part of OSG) and implements the last part of the synchronization described earlier. The server makes sure all ready-to-swap messages have come and orders all the clients to swap, thereby rendering a frame. The overhead involved is kept low by performing the communication between the client and the server in different threads and sometimes processes.

Garuda's ability to transparently render any OSG-based application brings clustered, tiled-display to a large number of applications. The user does not need to learn a new API or modify their programs in anyway. All aspects are handled implicitly if OSG is used in a standard manner. Garuda today supports OSG-based applications that use groups, geodes and static and dynamic transformation nodes. Support

for textures, normals and lighting is also present. Advanced features such as animation, FX effects, particles, etc., are not currently handled.

5.6 Experimental Results

The Garuda system was evaluated for its performance on typical complex scenes, and on its scalability to large displays. We want to achieve good frame rates on challenging applications. We also want to provide a scalable design such that very large display walls can be built using commodity hardware.

Test Setup: The test setup consists of 16 low-end systems with AMD Athlon 64 3000+ CPU, 512 MB RAM, and an ATI Radeon Xpress 200 on-board graphics. The GPU on these systems uses 64 MB of system RAM as the video memory. These machines act as rendering nodes in the cluster. For all experiments, each tile renders its frustum to a 1024×768 display in all configurations. The 4×4 display wall thus has a total resolution of 4096×3072, or well over 12 Million pixels. The server has an AMD Athlon 64 3200+ system with 3GB RAM. It has an Nvidia 6600GT GPU, but the graphics capabilities of the server are not important. Our server also doubles as a rendering node in our cluster. The low-end systems serve our objective of building a display wall using low-end hardware. An increase in the rendering capabilities at the clients results in a direct performance improvement for our system, as shown by the experiments below. Unless mentioned otherwise, the experiments are carried out over a 100 Mbps Ethernet network. We present results of tests for scalability with various tile-configurations (2×2, 2×3, 2×4, 3×3, 3×4, 4×4).

Two hierarchical OSG models were used for the experiments. The first is the model of Fatehpur Sikri[2], an architectural monument with a scenegraph hierarchy with $530,000$ triangles spread over 1113 geometry nodes with 331 internal nodes. Its average branching factor is 4.36. The second is a hierarchical version of the UNC Powerplant model. It consists of about 13 million triangles spread over 19666 geometry nodes with 4760 internal nodes. Its average branching factor is 5.13. The spatial hierarchy was created from the original non-hierarchical models which resulted better scene management at the cost of increase in the number of triangles. Another synthetic, hierarchical teapot model was also employed in testing certain aspects of the system as described later.

The server reads and initializes large amounts of data and sends a lot of it to the clients on system start up as described before. Table 5.1 compares the startup transmission time when using UDP multicast with the TCP unicast approach, for the Powerplant model. This includes the preprocessing of the scenegraph and the preparation and transmission of the data for the first frame. The startup time of our system is virtually independent of the tile-configuration when using multicast but increases linearly when using TCP unicast since it ends up sending the same object multiple times to each node that needs it. It should be noted that as the number of tiles go up, the frustum of each tile shrinks, and the num-

[2]Fatehpur Sikri is a world heritage site located near Agra. It was constructed by the Moghul emperor Akbar in the late 16^{th} century. The graphics model was created by NCST/CDAC, India.

Tile configuration	Time for TCP unicast (seconds)	Time for UDP multicast (seconds)
2×2	7.24	11.95
2×3	10.35	11.76
2×4	13.33	11.53
3×3	15.07	11.87
3×4	19.38	11.52
4×4	25.09	11.58

Table 5.1 Startup transmission times using the unicast and multicast approaches for the Powerplant model, including the transmission of the initial data to the clients. The time for multicast remains constant practically while the unicast time grows linearly with the number of tiles.

Tile configuration	Powerplant Culling time (ms)	Fatehpur Sikri Culling time (ms)
2×2	0.81	1.05
2×3	1.20	1.55
2×4	1.44	1.77
3×3	1.68	2.22
3×4	2.05	2.48
4×4	2.36	2.60

Table 5.2 Visibility determination time using the adaptive visibility culling algorithm on the full Powerplant and Fatehpur Sikri models.

ber of objects that are visible to multiple tiles increases. This results in greater gains for the multicast approach. This makes the Garuda architecture scalable to large display sizes.

The adaptive visibility algorithm is an important step of the display wall. Table 5.2 shows the time taken to perform the visibility determination on the Powerplant and Fatehpur Sikri models for various tile-configurations using Algorithm 9. The culling time increases sub-linearly as the number of tiles increases, owing to the hierarchical treatment of scene and frustums in our algorithm. This implies that the algorithm scales well for large display configurations. Sub-linear increase in the culling time improves the scalability of the system to a large number of tiles. The table shows the culling time averaged over a 3000 and 2700 frame typical walkthrough for the Powerplant and Fatehpur Sikri models respectively.

Figure 5.9 shows the performance of Garuda on a 2700-frame walkthrough on the Fatehpur Sikri model, that goes over dense and sparse areas of the model. In the dense regions, the performance is about 10 frames per second (fps) due to the limited rendering capabilities of the clients. Framerate increases to about 40 when less geometry is visible (frames 1000–1400 and 2100–2350). Note that the framerate remains constant for different configurations as the resolution increases four-folds from

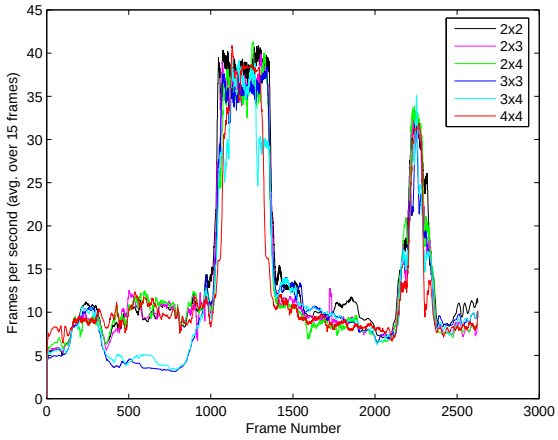


Figure 5.9 Framerate achieved during a 2700 frame walkthrough of the Fatehpur Sikri model. The performance remains almost unchanged for different tile-configurations though the display resolution increases four-fold.

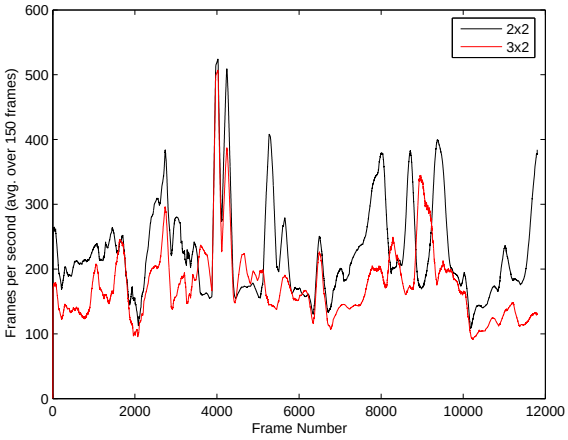


Figure 5.10 Framerate for a 12000 frame walkthrough of the Fatehpur Sikri model using high-end rendering nodes with Nvidia 6600GT graphics cards. The system achieves a rendering performance as high as 200 fps due to the improved graphics capabilities.

2×2 to 4×4. The multicast communication scheme and the server's data-push philosophy make it possible by facilitating concurrent transmission to multiple clients. The network requirements are kept to a minimum with the use of caching of objects at the clients.

The performance of the Garuda system is determined by the worst-performing rendering node. Better system performance can be obtained if better rendering performance is available at each tile. Figures 5.10 shows the results using high-end client nodes with 1GB RAM and Nvidia 6600GT GPU. The walkthrough used is more challenging than the previous experiment with viewpoints that see more of the model at all times. Frame rates of 200 fps and above were obtained, confirming that the experiment shown on Figure 5.9 is rendering limited on the low-end clients. Note the improvement in performance even using smaller tile configurations.

The fps does not increase with the number of tiles in Figures 5.9 and 5.10 because of the poor load distribution. A lot of the model geometry is concentrated in the lower quarter of the display (refer Figure 5.18) around the ground region. The ground region also contains models with large triangles which do not get frustum-culled effectively. The worst-performing rendering node ends up drawing about the same amount of scene even after the tile size reduces. Better load distribution can be observed for scenes with granularity such that the geometry is distributed evenly over the entire scene. Figure 5.11 shows a synthetic environment created to test this. The scene has 417 teapots with a total of 1.5 million triangles, that are distributed randomly in space. The figure shows near-linear increase in the frame rates when rendering to tile configurations of 2×2, 3×3, and 4×4. Based on these experiments, we can recommend that the the lower portions of a tiled display should perhaps be powered by nodes with high-end graphics for better performance on architectural models. Using different levels of detail (LoD) for

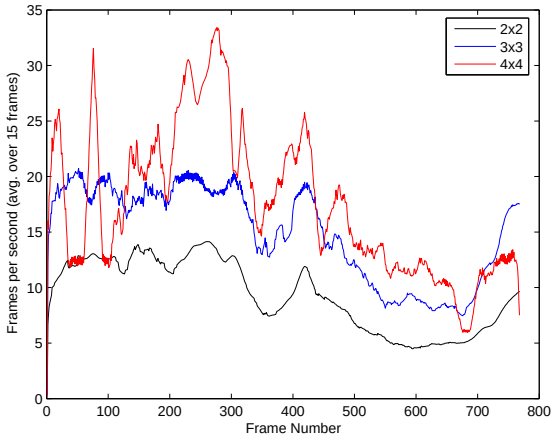


Figure 5.11 Rendering performance of Garuda on a synthetic environment with 417 randomly distributed teapots and 1.5 million triangles. The distributed rendering results in increased fps as the number of tiles increases on low-end clients.

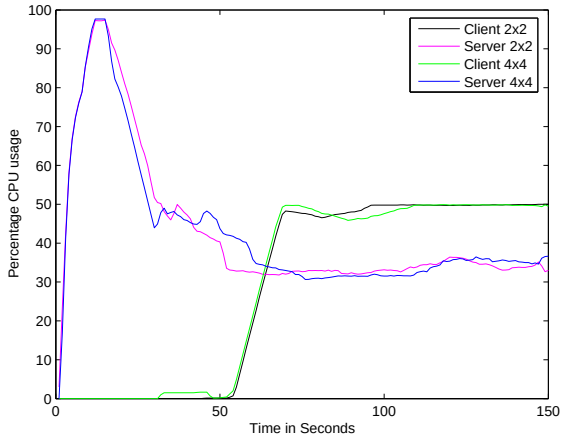


Figure 5.12 The CPU load for different tile sizes on the server and the client for the Powerplant model. CPU usage is nearly constant for different tile configurations resulting in excellent scalability.

objects is another option to ensure more even load distribution. This has to be done for the whole scene as all tiles should have the same LoD for an object. The LoD switching cannot easily be performed without the user application being aware of it. Since applicability to any OSG-based application was in initial goal, we did not pursue this approach further.

Figure 5.12 shows the CPU usage for different tile configurations at the server and a typical client node for Garuda. The startup phase is compute-intensive on the server, but its load settles to a comfortable 40% of CPU usage very quickly. The client's CPU usage picks up slowly since it only has rendering to perform. The client node's usage settles at a 50% level to manage the scenegraph, the cache and the rendering. The CPU usage is practically constant for a 2×2 configuration and a 4×4 configuration. Please note that the 50 seconds of startup time includes the time to load the models from files, to pre-process the loaded scenegraph, to compute the visibility for the first frame and to send the data to the clients. The loading time was not shown in Table 5.1. This again establishes that our system architecture is scalable to a large number of tiles.

Figure 5.13 demonstrates the low dependence of the system on the network bandwidth for a 4×4 configuration. The faster network performs better initially, as more and more scenegraph is discovered and fresh objects are transmitted to the clients (till frame number 900). Both 10 Mbps and 100 Mbps networks perform equally well thereafter. This is due to the caching of geometry at the tiles, which ensures minimal transmission of bulky geometry data for subsequent frames. These results suggest that the Garuda architecture can comfortably scale to display walls that have several dozens of tiles using commodity LANs of 100 or 1000 Mbps.

We show the impact of caching for different cache sizes in Figure 5.14 for a 2×2 display wall. A 2×2 wall was used here as cache size is more critical for smaller tile configurations. The experiment

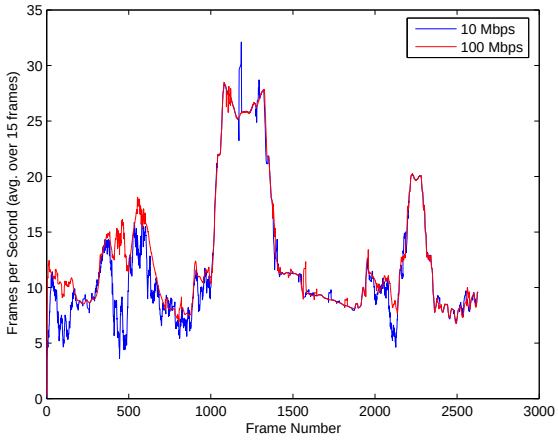


Figure 5.13 Framerates for the 2700 frame walkthrough of the Fatehpur Sikri model on a 4×4 wall with 10 Mbps and 100 Mbps networks. Caching at clients significantly lowers the network dependence, both networks have near-constant fps after the initial startup time.

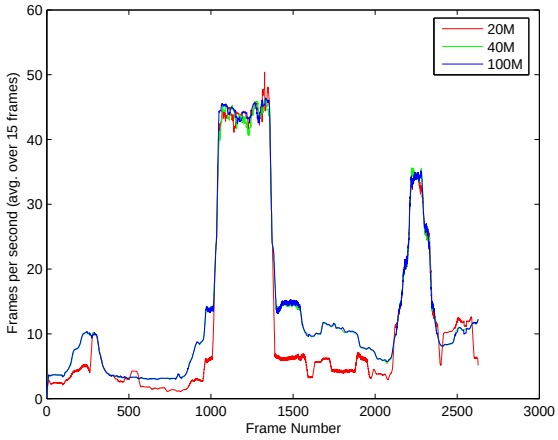


Figure 5.14 Framerates for a 2×2 system for three cache sizes on the Fatehpur Sikri model for the 2700 frame walkthrough. 20 MB cache has lesser frame rate because of retransmission of geometry data.

shows the same walkthrough as in Figure 5.9 for three different cache sizes. The optimal cache size depends on the density of objects in an environment and can vary from scene to scene. It can be seen that the framerates are less with a 20 MB cache than with 40 MB or 100 MB cache. This is due to the retransmission of objects necessitated by their eviction when cache fills up. For the regions with higher framerates (frames 1000–1200 and 2100–2300) there is lesser geometry to render in the walkthrough. No retransmission is required even with smaller caches. The framerates for the 40 MB and 100 MB are almost identical, which suggests that a 40 MB cache is sufficient for this model. It is interesting to consider the case when the cache is sufficient to hold the whole scene at each rendering client. The performance of Garuda's client-server architecture will approach that of the master-slave architecture [32], but with better handling of dynamic objects. The server needn't perform elaborate visibility computations. This, however, will not be compatible with our initial design goal of using an inexpensive processor as the rendering client.

Figure 5.15 shows Garuda's performance on a walkthrough of the Powerplant model. All tiles are powered by high-end systems with an Nvidia 6600GT card in each. The framerate is maintained above 40 fps for most of the walkthrough even on such a challenging model. For comparison, we setup a 2×2 configuration involving the high-end machines using Chromium for tiled rendering. The system was able to achieve only about 0.2 fps due to the high network requirements. The same walkthrough performs at about 10 fps on a single machine of similar configuration. The display wall achieves a framerate $4 - 5$ times higher in spite of increasing the pixel resolution 4 to 6 times. The high variability in the fps is due to the inherent randomness associated with network communications while the overall improvement in fps is due to the cluster rendering.

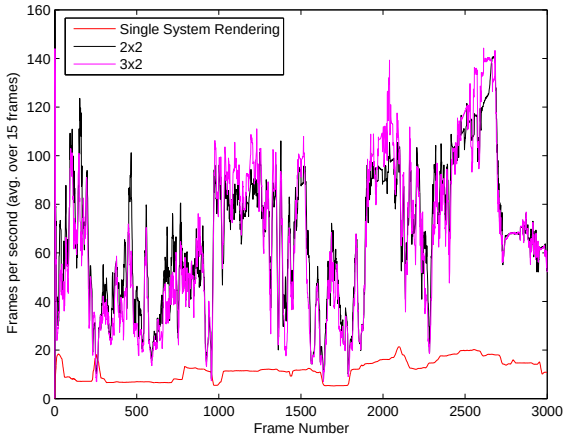


Figure 5.15 Framerate for an 800 frame walkthrough of the Powerplant model using rendering nodes with Nvidia 6600GT graphics. The system performs at more than 40 fps for most part of the walkthrough. The 3×2 configuration performs better due to lower rendering load on individual tiles.

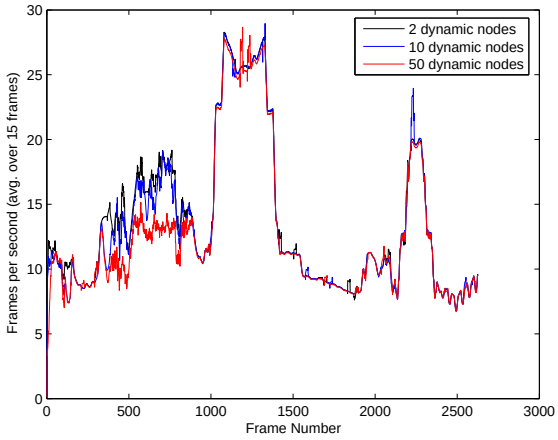


Figure 5.16 Fatehpur Sikri walkthrough for a 4×4 system with dynamic Open Scene Graph transformation nodes. The scene has different number of rotating cow models (each with 5800 triangles) added to it. The transformation nodes are updated every frame. The fps remains constant practically. Low-end clients were used for this test.

The same walkthrough runs on a 4×4 configuration using the low-end clients (ATI Xpress 200 with 64 MB shared video memory) at about 3 fps. The application fails to startup due to poor resources on a single system of same capabilities. This demonstrates that the Garuda system is able to exploit the power of distributed rendering to render challenging models using modest computers.

Dynamic scenes can also be rendered at interactive frame rates using the Garuda system. Figure 5.16 shows the system's performance for a scene with many dynamic objects. The Fatehpur Sikri model was modified by adding 2, 10 and 50 Open Scene graph dynamic transformation nodes, each with a cow model of 5800 triangles. Data for these nodes changes every frame and hence must be sent every frame, adding to the the overhead of rendering dynamic environments. Culling at each frame needs to recompute the transformed OBB for each node for proper results. The system is able to maintain more than 10 FPS throughout the walkthrough even with large number of moving objects in the scene. The graph also shows that Garuda is scalable with respect to number of dynamic nodes in the scene; as the reduction in fps is very low even for a large increase in dynamic nodes.

We also check to see to what extent a single server can be pushed to power a tiled display. Figure 5.17 shows a single server culling time on the Fatehpur Sikri model for the 2700 frame walkthrough for various tile sizes. The culling time increases sub-linearly with the increase in tile size which adds to scalability of the Garuda system to large configurations. Although the increase is sub-linear, for an 8×8 tile configuration a single server takes around 6.5 ms for culling only which leaves around 9 ms for other overheads and data transmission if the required 60 fps is expected. Clearly for a larger display system a hierarchy of servers will be need, consisting of culling stages. First stage culls to a bigger

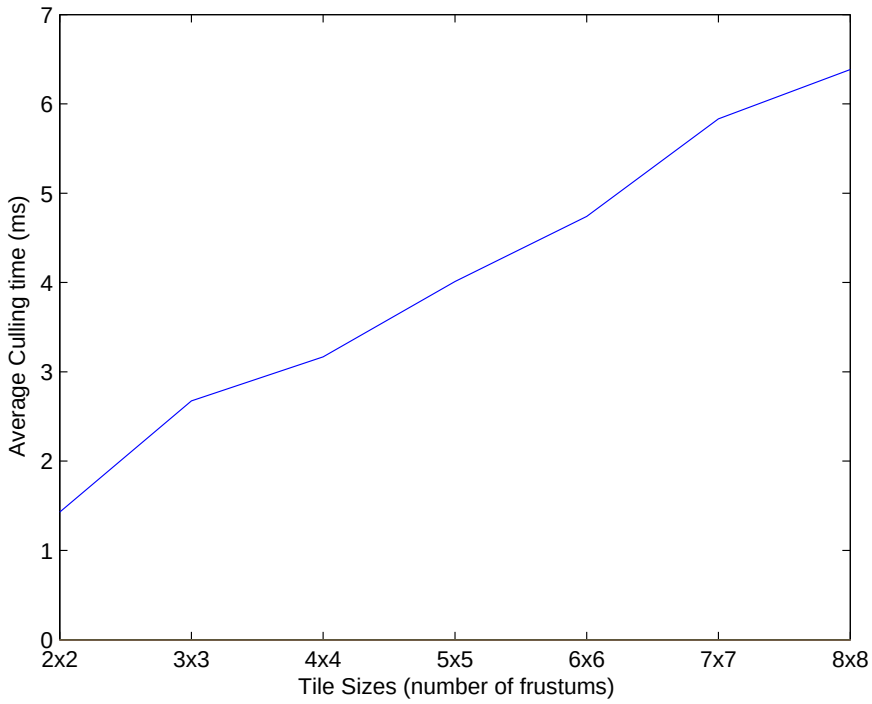


Figure 5.17 Fatehpur Sikri 2700 frame walkthrough's average culling times. The increase in culling time is sub-linear, adding to the scalability of the Garuda system. Also showing that the culling becomes a bottleneck after 7×7 tile configuration for a single server system.

sub-frustum and sends the information to next stage which in turn culls to the tile frustums inside its sub-frustum. The second stage is done in parallel and hence much culling time is saved. A single server hence becomes a bottleneck after a 7×7 tile configuration.

5.7 Conclusions and Future Work

In this chapter, we presented Garuda, a geometry-managed, tiled display system built using commodity computers that can transparently render any application built using the Open Scene Graph API. The design choices made with respect to the hardware and software make Garuda scalable to large displays. It can also find wide applications due to its ability to render any OSG-based application without modifying it. Local caching of the geometry and the multicast mode of transmission keeps the network requirements moderate and the system scalable. Garuda was shown to be scalable for different tile geometry using commodity components. Spatial and temporal coherence are exploited by using a

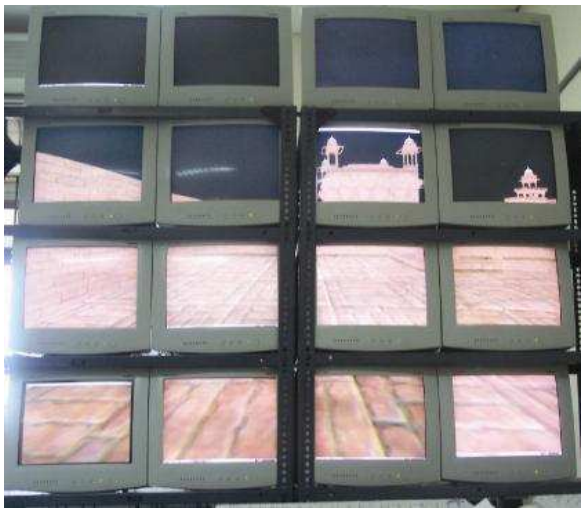


Figure 5.18 A 4×4 display wall showing the Fatehpur Sikri model. The combined resolution is 12 mega pixels.

Figure 5.19 A 4×4 display wall showing the Powerplant model. The combined resolution is 12 mega pixels.

push-philosophy adopted by the server. This also allows for consistent handling of dynamic objects in the scenegraph.

Swap-lock and gen-lock of the display system need further research. Since the use of commodity graphics cards is a design goal of our system, hardware gen-lock is not an option. We are working on a camera based scheme to bring the displays into perfect synchronization. A high-speed camera and a calibration pattern that appears on all tiles in the same frame can together estimate the delay between each tile's display from a reference tile. It is possible to adjust the low-level video parameters of the display card till the delay is minimized. This may be performed in an iterative manner till the genlocks are sufficiently synchronized. This needs further experimentation.

Chapter 6

Conclusions

In this thesis, we presented ways to utilize computation in order to enhance displays. These enhancements take into consideration the physical processes involved in the system and use this prior knowledge to produce enhanced experiences that are not produceable directly using conventional displays. We designed and implemented four systems to this end, each dealing with an independent aspect of the display design problem set.

We prototyped a complete pipeline that produces a precise and accurate rendering of 3D environments for a single head tracked viewer on any *multi-planar display*. We demonstrated how it scales with various parameters both in simulation and on actual displays we built. Scalability across multiple form factors was also demonstrated. We studied the applicability of such a display in comparison to existing displays and also compared our rendering approach with multiple existing methods. We concluded that our method outperforms existing methods in terms of visual quality and speed. Numerous applications benefit from such a display and can help users better visualize his/her design - especially in the medical, mechanical, and CAD/CAM domains - where quality renderings are necessary.

Next, we explored another dimension of the same problem in *parametric displays*, where the display shape is defined exactly using parametric or implicit equations. By bypassing the two pass texture mapping previously employed for curved displays our approach retains pixel density and hence quality. The method produces quality renderings in real-time using the tessellation hardware of current generation GPUs. We demonstrated that though an approximate rendering system, the display is capable of producing high quality images for any parametric shape given a mapping to a planar domain. In applications where speed takes precedence over quality, such as games, parametric displays can produce truly immersive experiences.

Given the same hypothesis of computational displays, we explored different problems where such an approach can benefit user experiences. We enhanced the intensity resolution of a given display in Chapter 4 by sacrificing its spatial and temporal characteristics. This led to a better granularity of the contrast already present on the base display, in turn providing the user with better dynamic range that can be perceived directly by the human eye. Such a display has tremendous applications in visualization and medical domains. Given a high bit data (which is common in many domains) a user can now map

the data on to a given display without losing information. Decision making based on images can thus be improved, providing better diagnosis in medial and better analysis for visualization applications.

Lastly, in Chapter 5 we explored the important yet different problem of focus+context using the same computational display hypothesis. We demonstrated that our system can render and display massive environments at interactive frame rates to an arbitrary configuration of tiled displays, creating a large context while at the same time retaining pixel density. Our system uses existing graphics framework and can port any application without needing to modify the source code, increasing its applicability. The system's capabilities across various factors were studied and evaluated. In conclusion we stated that the design principles we opted scale with various configurations and that increasing the display size results in better performance. Visualization applications where detail is required while viewing its context, such as in biological simulations and fluid mechanics, our system provides an off-the-shelf solution without compromising focus, context or interactivity of the application.

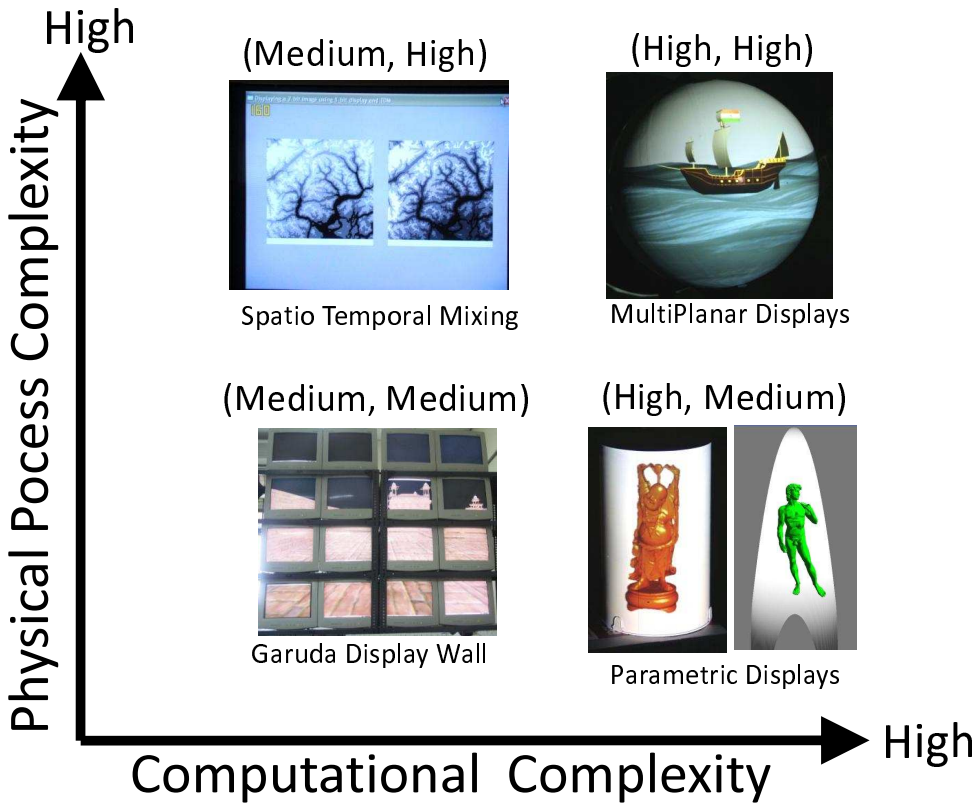


Figure 6.1 Various display systems we proposed in this thesis shown in terms of physical process and computational complexities involved.

The systems presented in this thesis all followed the computational displays framework. The systems did not rely on any specific display technology and can be applied to any display technology, making our solutions scalable. The hypothesis we considered in this thesis is to use computation to enhance

displays. This was applied to various unrelated problems in display design. In each, we applied the same framework to come to a problem-specific design. We then implemented this design using computation to produce the desired result. Each system highlighted various aspects of our computational display framework - using one or multiple of its attributes. In this process we explored single, parallel and distributed algorithmic load distribution along with various of their permutations. We also explored inter and intra display processes along with multiple and single display elements in various systems and were able to use the same framework for such a varying set of problem.

The set of problems explored in this thesis are by no means an exhaustive study of the framework we propose. They do, however, underline a common theme amongst radically different requirements. Pictorially the area covered by these problems is shown in Figure 6.1. The problems are segregated in terms of their physical process complexity vs computational complexity. As seen, these problems cover a suitable ground in our framework - with both physical complexity and computational complexity ranging from medium to high. We were able to solve this range of problems with different requirements using the same hypothesis. This validates our hypothesis by example and also reinforces the idea of using computation in display design as much as other means. These problems, however, do not begin to scratch the surface of the subject that is Computational Displays.

In our problems, we used the underlying display technology as a black box with limited hardware modification. If more focus could be given on to hardware modifications, even better visual experiences could be achieved, such as auto-stereoscopy, holography, volumetric, True 3D, and even the ultimate display [114]. These problems along with others are worth studying to concertize the idea of display enhancement using computation.

In this thesis, we explored one niche of the computational displays landscape, one where computation is heavily used in conjunction with simple physical processes. A direct technical approach may also produce similar experiences given enough budget and time. Our systems, however, provided these experiences at a price that is significantly less, using off-the-shelf hardware. The ideas we explore in this thesis can be extended on to hardware modifications involving optics, mechanical and metallurgical elements. The underlying design, however, should always respect the role of computation and what can be achieved with it when used intelligently.

Future work can directly use the computational display framework to explore various other problems. This will amount to solving the more niche landscape explored in this thesis, as stated before, this can be claimed as a subset of the larger Computational Displays subject, termed as *Computationally Enhanced Displays*. Work in this area can explore problems such as increasing the vertical refresh of a given display. Using multiple overlaid images, either using projection or LCDs with optical elements such as half-mirrors, the images can be controlled using off-the-shelf hardware, such as Nvidia shutter glasses or delay circuits. An image can then be shown while shutting off images from other sources, collectively producing a higher refresh rate in the observer's eyes.

Another area the framework can be applied to is haptics. The idea is to provide feedback to the user. Using an FTVR setup on a single plane connected to, say, an array of electromagnets at each

pixel beneath it, we can *virtually project* the depths of the scene into heights by increasing or decreasing the strength of each electromagnet. Depth can then be *touched* by the observer wearing an oppositely charged magnetic glove, thus enabling the user to feel as well as see the height of the projected scene.

As said earlier, using other means such as optics, mechanical and metallurgical means along with computation one can attempt even bigger problems in display design. This can be applied inside the display to modify it at a hardware level, thus falling beyond what is covered in this thesis, however, still under the ambit of Computational displays.

One example of such a future effort would be to create a true 3D display. The idea is to light a voxel in space, or rather a beam of voxels in space along a line, such that they can be viewed from any angle. This can be done using amplitude modulation of laser beams. We know from Fourier Analysis that any given signal can be represented by a sum of sine and cosine waves. The idea is, given a beam of light in space, a 1D signal can be specified such that its highs correspond to specific intensities of voxels along that beam. Decomposing the signal using Fourier Analysis will yield a set of sine and cosine waves, whose superimposition (addition) in air necessarily yields the same signal. The decomposed sine and cosine waves can be used to amplitude modulate lasers of a given wavelength. And combining them together along a principal axis will ensure interference, since the carrier, laser, is same for all decomposed waves. One thing to consider in this design is the speed of the carrier (speed of light). The design works if the waves are static in space, however, falls apart if they move at the same speed (given the laser as the carrier this is always the case). Initial experimentation on this design shows that a phase difference is necessary in order to create the desired effect - this directly translates to some slower moving waves than others. Numerous physicists have controlled the speed of light in many mediums using off-the-shelf hardware [47]. Increasing or decreasing the speed by a factor can simulate the phase difference, thus allowing individual waves to interfere correctly and reproduce the given pattern in air. Multiple such beams can be produced using time division multiplexing to cover a volume in space, yielding a true 3D display. More analysis and experimentation is needed to perfect the design and is currently underway by the author of this thesis.

Related Publications

- Pawan Harish and P. J. Narayanan. ``Designing Prespectively-Correct MultiPlanar Displays´´. In IEEE Transactions on Visualization and Computer Graphics, 2013. TVCG Vol 19, No. 3, Pages 407-419, March 2013.
- Pawan Harish and P. J. Narayanan.``A View Dependent, Polyhedral 3D Display´´, in ACM SIGGRAPH Virtual Reality Continuum and its Applications in Industry 2009. Pages 71-75.
- Pawan Harish and P. J. Narayanan. ``View Dependent Rendering to Simple Parametric Display Surfaces´´. In EUROGRAPHICS Joint Virtual Reality Conference of EuroVR-EGVE, 2011. Pages 27-30.
- Pawan Harish, Parikshit Sakurikar and P. J. Narayanan. ``Spatio-Temporal Mixing to Increase Intensity Resolution on a Single Display´´, In CVPR Workshop on Computational Cameras and Displays, 2012.
- Pawan Harish, Parikshit Sakurikar and P. J. Narayanan. ``Increasing Intensity Resolution on a Single Display using Spatio-Temporal Mixing´´, In Indian Conference for Computer Vision, Graphics and Image Processing, 2012
- Nirnimesh, Pawan Harish and P. J. Narayanan. ``Garuda: A Scalable Tiled Display Wall using Commodity PCs´´. In IEEE Transactions on Visualization and Computer Graphics, 2007. TVCG Vol 13, No.5, Pages 864-877, Sep/Oct 2007.
- Nirnimesh, Pawan Harish and P. J. Narayanan. ``Culling an Object Hierarchy to a Frustum Hierarchy´´. In Indian Conference on Vision, Graphics and Image Processing, 2006. LNCS 4338, Pages 252-263.

Bibliography

[1] BrightSide DR37-P HDR display, http://www.bit-tech.net/hardware/2005/10/03/brightside_hdr_edr/1 .

[2] Distributed Multihead X Project (DMX) http://dmx.sourceforge.net.

[3] GigaPixel Project, Virginia Tech http://infovis.cs.vt.edu/gigapixel/index.html.

[4] HIPerWall, http://cg.calit2.uci.edu/mediawiki/index.php.

[5] JVC Reference series 3D Home Cinema 4K Projector http://pro.jvc.com/prof/attributes/tech_desc.jsp?model_id=MDL102123&feature_id=02.

[6] Light And Color http://www.colorado.edu/physics/phys1230/phys1230_sp09/classnotes/15_Vision_Time_and_Space.pdf.

[7] LionEyes Display Wall. Penn State University. http://viz.aset.psu.edu/ga5in/DisplayWall.html.

[8] ModViz Renderizer, http://www.modviz.com/products/renderizer.asp.

[9] OSG: OpenSceneGraph. http://www.openscenegraph.org.

[10] PowerWall. University of Minnesota. http://www.lcse.umn.edu/research/powerwall/powerwall.html.

[11] VisWall High Resolution Display Wall. http://www.visbox.com/wallMain.html.

[12] Artoolkit www.hitl.washington.edu/artoolkit/, 2002.

[13] AIST. Three Dimensional Images in the Air. www.aist.go.jp/aist_e/latest_research/2006/20060210/20060210.html, 2006.

[14] K. Akeley and J. Su. Minimum triangle separation for correct z-buffer occlusion. In *ACM SIGGRAPH/EUROGRAPHICS GH '06*, pages 27–30, 2006.

[15] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Component Cluster. In *IEEE Virtual Reality Conference*, pages 273–274, 2002.

[16] W. Allen and R. Ulichney. Wobulation: Doubling the addressed resolution of projection displays. *Symposium on Information Displays*, 2005.

[17] M. Ashdown, M. Flagg, R. Sukthankar, and J. M. Rehg. A flexible projector-camera system for multi-planar displays. In *EEE CVPR*, pages 165–172, 2004.

[18] P. Baudisch, N. Good, and P. Stewart. Focus plus context screens: combining display technology with visualization techniques. In *ACM UIST*, UIST '01, pages 31–40, 2001.

[19] P. Baudisch, N. Good, and P. Stewart. Focus plus context screens: Combining display technology with visualization techniques. In *ACM Symposium on User Interface Software and Technology*, 2001.

[20] H. Benko, R. Jota, and A. Wilson. Miragetable: freehand interaction on a projected augmented reality tabletop. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 199–208, New York, NY, USA, 2012. ACM.

[21] H. Benko, A. D. Wilson, and R. Balakrishnan. Sphere: multi-touch interactions on a spherical display. In *UIST*, UIST '08, pages 77–86, 2008.

[22] F. Berthouzoz and R. Fattal. Resolution enhancement by vibrating displays. *ACM Trans. Graph.*, 31(2):15:1–15:14, Apr. 2012.

[23] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: a virtual platform for virtual reality application development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)*, pages 89–96, Washington, DC, USA, 2001. IEEE Computer Society.

[24] O. Bimber, G. Wetzstein, A. Emmerling, and C. Nitschke. Enabling view-dependent stereoscopic projection in real environments. In *IEEE/ACM ISMAR*, ISMAR '05, pages 14–23, 2005.

[25] P. Bodrogi, K. Muray, and J. Schanda. Accurate colorimetric calibration of crt monitors. In *SID*, page 455, 1995.

[26] K. R. Boff, L. Kaufman, and J. P. Thomas. *Handbook of perception and human performance*, volume 1, pages 13–1 to 13–62. 1986.

[27] P. Bourke. idome: Immersive gaming with the unity game engine. In *CGAT 09*, CGAT '09, pages 265–272, 2009.

[28] T. C. Bressoud and F. B. Schneider. Hypervisor-based fault-tolerance. In *Fifteenth ACM Symposium on Operating System Principles (ASPLOS V)*, pages 1–11, Copper Mountain Resort. Colorado, 1995. ACM.

[29] M. Brown, A. Majumder, and R. Yang. Camera-based calibration techniques for seamless multiprojector displays. *Visualization and Computer Graphics, IEEE Transactions on*, 11(2):193–206, March-April.

[30] M. S. Brown and A. Majumder. SIGGRAPH 2003 Course Notes: Large-Scale Displays for the Masses, 2003.

[31] K. K. Charles Hains, Shen-Ge Wang. Chapter 6, digital color halftones. In *Digital Color Imaging Handbook*. Xerox Corporation, 2003.

[32] H. Chen, D. W. Clark, Z. Liu, G. Wallace, K. Li, and Y. Chen. Software environments for cluster-based display systems. In *IEEE International Symposium on Cluster Computing and the Grid*, 2001.

[33] J. Cohen, C. Tchou, T. Hawkins, and P. E. Debevec. Real-time high dynamic range texture mapping. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 313–320. Springer-Verlag, 2001.

[34] W. T. Corrêa, J. T. Klosowski, and C. T. Silva. Visibility-Based Prefetching for Interactive Out-Of-Core Rendering. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2003.

[35] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93*, 1993.

[36] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH*, pages 135–142, 1993.

[37] P. E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 369–378, 1997.

[38] M. Deering. High resolution virtual reality. *SIGGRAPH Comput. Graph.*, 26(2):195–202, 1992.

[39] M. Deering and D. Naegle. The SAGE graphics architecture. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 683–692, New York, NY, USA, 2002. ACM Press.

[40] J. Djajadiningrat, C. Overbeeke, and P. Stappers. Cubby: A unified interaction space for precision manipulation. In *ITEC 2001*, ITEC 2001, pages 24–26, 2001.

[41] N. A. Dodgson, J. R. Moore, and S. R. Lang. Multi-view autostereoscopic 3d display. *IEEE Computer*, 38:31–36, 1999.

[42] G. E. Favalora, J. Napoli, D. M. Hall, R. K. Dorval, M. Giovinco, M. J. Richmond, and W. S. Chun. 100 million voxel volumetric display. In *Proceedings of the SPIE*, volume 4712, pages 300–312, 2002.

[43] S. Fisher. Viewpoint dependent imaging: An interactive stereoscopic display. *SPIE*, page 367, 1982.

[44] J. W. Frens, J. P. Djajadiningrat, and C. J. Overbeeke. Cubby+: exploring interaction. In *DIS*, DIS '02, pages 135–140, 2002.

[45] J. Gao, J. Huang, C. R. Johnson, S. Atchley, and J. A. Kohl. Distributed data management for large volume visualization. In *IEEE Visualization*, page 24, 2005.

[46] D. Germans, H. J. Spoelder, L. Renambot, and H. E. Bal. VIRPI: A High-Level toolkit for interactive scientific visualization in virtual reality. In *Immersive Projection Technology / Eurographics Virtual Environments Workshop*, pages 109–120, 2001.

[47] M. Gonzlez-Herrez, K.-Y. Song, and L. E. Thvenaz. http://physicsworld.com/cws/article/news/2005/aug/23/fibres-control-the-speed-of-light, 2005.

[48] Y. H. H. Kikuchi, H. Higuchi and T. Iwata. Fast electro-optical switching in polymer-stabilized liquid crystalline blue phases for display application. In *SID Int. Symp. Digest Tech*, pages 1737–1740, 2007.

[49] U. Hahne, J. Schild, S. Elstner, and M. Alexa. Multi-touch focus+context sketch-based interaction. In *Eurographics SBIM*, SBIM '09, pages 77–83, 2009.

[50] P. Harish and P. J. Narayanan. A view-dependent, polyhedral 3d display. In *VRCAI*, VRCAI '09, pages 71–75, 2009.

[51] C. Harrison, H. Benko, and A. D. Wilson. Omnitouch: wearable multitouch interaction everywhere. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 441–450, New York, NY, USA, 2011. ACM.

[52] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[53] E. F. Helga Kolb, Ralph Nelson and B. Jones, editors. *WebVision: The Organization of the Retina and Visual System*. University of Utah, 1995.

[54] X. Hou, L.-Y. Wei, H.-Y. Shum, and B. Guo. Real-time multi-perspective rendering on graphics hardware. In *EUROGRAPHICS Symposium on Rendering*, EGSR '06, pages 93–102, 2006.

[55] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702, New York, NY, USA, 2002. ACM Press.

[56] H. Iwata. Full-surround image display technologies. *Int. J. Comput. Vision*, 58:227–235, 2004.

[57] G. James and J. ORorke. Real-time glow. In *GPU Gems*, pages 343–362. 2004.

[58] A. Jones, I. McDowall, H. Yamada, M. T. Bolas, and P. E. Debevec. Rendering for an interactive 360degree light field display. *ACM Trans. Graph.*, 26(3), 2007.

[59] S. Kettner, C. Madden, and R. Ziegler. Direct rotational interaction with a spherical projection. In *In Interaction: Systems, Practice and Theory Proceedings*, 2004.

[60] A. G. Kirk and J. F. O'Brien. Perceptually based tone mapping for low-light conditions. *ACM Trans. Graph.*, 30(4):42:1–42:10, Aug. 2011.

[61] G. Krawczyk, K. Myszkowski, and H.-P. Seidel. Lightness perception in tone reproduction for high dynamic range images. In *The European Association for Computer Graphics 26th Annual Conference EUROGRAPHICS 2005*, volume 24, pages 635–645, 2005.

[62] A. Kulik, A. Kunert, S. Beck, R. Reichel, R. Blach, A. Zink, and B. Froehlich. C1x6: a stereoscopic six-user display for co-located collaboration in shared virtual environments. *ACM Trans. Graph.*, 30(6):188:1–188:12, Dec. 2011.

[63] S. Kuthirummal, H. Nagahara, C. Zhou, and S. K. Nayar. Flexible depth of field photography. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:58–71, 2011.

[64] D. Lanman, M. Hirsch, Y. Kim, and R. Raskar. Content-adaptive parallax barriers: optimizing dual-layer 3d displays using low-rank light field factorization. *ACM Trans. Graph.*, 29(6):163:1–163:10, Dec. 2010.

[65] A. Law, D. Aliaga, B. Sajadi, A. Majumder, and Z. Pizlo. Perceptually-based appearance modification for compliant appearance editing. *Computer Graphics Forum*, 2011.

[66] P. Ledda, A. Chalmers, T. Troscianko, and H. Seetzen. Evaluation of tone mapping operators using a high dynamic range display. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 640–648, New York, NY, USA, 2005. ACM.

[67] M. Levoy, B. Chen, V. Vaish, M. Horowitz, I. McDowall, and M. Bolas. Synthetic aperture confocal imaging. *ACM Trans. Graph.*, 23(3):825–834, Aug. 2004.

[68] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM.

[69] S. Liao, M. A. Lopez, and D. Mehta. Constrained polygon transformations for incremental floorplanning. *ACM Trans. Des. Autom. Electron. Syst.*, 6:322–342, 2001.

[70] P. Lincoln, A. Nashel, A. Ilie, H. Towles, G. Welch, and H. Fuchs. Multi-view lenticular display for group teleconferencing. In *International Conference on Immersive Telecommunications*, IMMERSCOM '09, pages 22:1–22:8, 2009.

[71] G. Lippmann. Epreuves reversible donnant la sensation du relief. In *Journal of Physics*, volume 7, pages 821–825, 1908.

[72] R. Lopez-Gulliver, S. Yoshida, S. Yano, and N. Inoue. gCubik: a cubic autostereoscopic display for multiuser interaction: grasp and group-share virtual images. In *SIGGRAPH Posters*, page 133. ACM, 2008.

[73] A. Majumder. Contrast enhancement of multi-displays using human contrast sensitivity. In *CVPR'05*, CVPR '05, pages 377–382, 2005.

[74] A. Majumder and R. Stevens. Color nonuniformity in projection-based displays: Analysis and solutions. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):177–188, 2004.

[75] Microsoft. Microsoft Surface 2.0, http://www.microsoft.com/surface/, 2007.

[76] L. Moll, M. Shand, and A. Heirich. Sepia: Scalable 3d compositing using pci pamette. In *FCCM '99: Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 146–155, Washington, DC, USA, 1999. IEEE Computer Society.

[77] T. Möller. A fast triangle-triangle intersection test. *Journal of graphics, gpu, and game tools*, 2(2):25–30, 1997.

[78] M. Naef, E. Lamboray, O. Staadt, and M. Gross. The Blue-C distributed scene graph. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 125–133, New York, NY, USA, 2003. ACM Press.

[79] S. Nayar. Catadioptric omnidirectional camera. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 482 –488, jun 1997.

[80] S. K. Nayar and V. N. Anand. 3d display using passive optical scatterers. *IEEE Computer*, 40(7), 2007.

[81] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR '11*, pages 127–136. IEEE Computer Society, 2011.

[82] T. Ni, G. S. Schmidt, O. G. Staadt, M. A. Livingston, R. Ball, and R. May. A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In *Virtual Reality*, 2006.

[83] Nirnimesh, P. Harish, and P. Narayanan. Garuda: A scalable tiled display wall using commodity pcs. *IEEE TVCG*, 13:864–877, 2007.

[84] Nirnimesh, P. Harish, and P. J. Narayanan. Culling an object hierarchy to a frustum hierarchy. In *Indian Conference on Computer Vision, Graphics and Image Processing*, volume 4338 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[85] Nvidia. CUDA Programming Guide for CUDA Toolkit 3.2, 2011.

[86] S. Patidar and P. J. Narayanan. Scalable Split and Gather Primitives for the GPU. Technical Report IIIT/TR/2009/99, 2009.

[87] A. Pfahnl. Properties of fast-decay cathode ray tube phosphorus. In *Bell System Technical Journal*, v42, pages 181–201, 1963.

[88] R. Raskar. Immersive planar display using roughly aligned projectors. In *IEEE VR '00*, 2000.

[89] R. Raskar, K.-H. Tan, R. Feris, J. Yu, and M. Turk. Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 679–688, New York, NY, USA, 2004. ACM.

[90] R. Raskar, G. Welch, and H. Fuchs. Seamless projection overlaps using image warping and intensity blending. In *Virtual Systems and Multimedia*, 1998.

[91] R. Raskar, G. Welch, K.-L. Low, and D. Bandyopadhyay. Shader lamps: Animating real objects with image-based illumination. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 89–102. Springer-Verlag, 2001.

[92] P. Read, M. Meyer, and G. Group. *Restoration of Motion Picture Film*. Butterworth-Heinemann Series in Conservation and Museology. Butterworth-Heinemann, 2000.

[93] J. Rohlf and J. Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM Press.

[94] A. J. Sabri, R. G. Ball, A. Fabian, S. Bhatia, and C. North. High-resolution gaming: Interfaces, notifications, and the user experience. *Interact. Comput.*, 19:151–166, March 2007.

[95] B. Sajadi, M. Gopi, and A. Majumder. Edge-guided resolution enhancement in projectors via optical pixel sharing. *ACM Trans. Graph.*, 31(4):79, 2012.

[96] B. Sajadi, D. Lai, A. Ihler, M. Gopi, and A. Majumder. Image enhancement in projectors via optical pixel shift and overlay. *ACM Trans. Graph.*, 28(4):12, 2012.

[97] B. Sajadi and A. Majumder. Autocalibration of multiprojector cave-like immersive environments. *IEEE Trans. Vis. Comput. Graph.*, 18(3):381–393, 2012.

[98] D. Salomon. *Data Compression: The Complete Reference*. Second edition, 2000.

[99] R. Samanta, T. Funkhouser, and K. Li. Parallel rendering with k-way replication. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 75–84, 2001.

[100] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 97–108, 2000.

[101] T. A. Sandstrom, C. Henze, and C. Levit. The Hyperwall. In *CMV '03: Proceedings of the conference on Coordinated and Multiple Views In Exploratory Visualization*, pages 124–133, Washington, DC, USA, 2003. IEEE Computer Society.

[102] M. Sato, M. Inoue, M. Kasuga, and N. Hashimoto. Hdr display with a composite response function. In *Joint Virtual Reality Conference of EuroVR - EGVE posters*, JVRC 2011 posters, pages 86–88, 2011.

[103] B. Schaeffer and C. Goudeseune. Syzygy: Native PC Cluster VR. In *VR '03: Proceedings of the IEEE Virtual Reality 2003*, pages 15–22. IEEE Computer Society, 2003.

[104] S. Schkolne, M. Pruett, and P. Schröder. Surface drawing: creating organic 3d shapes with the hand and tangible tools. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, pages 261–268, 2001.

[105] H. Seetzen, W. Heidrich, W. Stuerzlinger, G. Ward, L. Whitehead, M. Trentacoste, A. Ghosh, and A. Vorozcovs. High dynamic range display systems. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 760–768, 2004.

[106] S. Sengupta, M. Harris, Y. Zhang, and O. J. D. Scan primitives for GPU computing. In *ACM SIGGRAPH/EUROGRAPHICS GH 07*, pages 97–106, 2007.

[107] L. Shupp, R. Ball, B. Yost, J. Booker, and C. North. Evaluation of viewport size and curvature of large, high-resolution displays. In *Proceedings of Graphics Interface 2006*, GI '06, pages 123–130, 2006.

[108] J. M. Singh and P. J. Narayanan. Real-time ray tracing of implicit surfaces on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 16:261–272, 2010.

[109] R. Singh, B. Jeong, L. Renambot, A. E. Johnson, and J. Leigh. Teravision: a distributed, scalable, high resolution graphics streaming system. In *CLUSTER*, pages 391–400, 2004.

[110] J. Son, B. Javidi, and K. Kwack. Methods for displaying three-dimensional images. In *Proceedings of the IEEE*, volume 94, pages 502–523, March 2006.

[111] Sony. RayModeler, blog.discover.sonystyle.com/ raymodeler-3d-prototype-will-be-showcased-at-siggraph, 2010.

[112] I. Stavness, B. Lam, and S. Fels. pcubee: a perspective-corrected handheld cubic display. In *Human factors in computing systems*, CHI '10, pages 1381–1390, 2010.

[113] I. Stavness, F. Vogt, and S. Fels. Cubee: a cubic 3D display for physics-based interaction. In *SIGGRAPH '06 Sketches*, page 165, 2006.

[114] I. E. Sutherland. The ultimate display. In *Proceedings of the Congress of the Internation Federation of Information Processing (IFIP)*, volume 2, pages 506–508, 1965.

[115] TrackIR5. NaturalPoint TrackIR5, www.naturalpoint.com/trackir/products/trackir5/, 2009.

[116] T. van der Schaaf, L. Renambot, D. Germans, H. Spoelder, and H. Bal. Retained mode parallel rendering for scalable tiled displays. In *Immersive Projection Technology (IPT) Symposium*, 2002.

[117] A. Veeraraghavan, R. Raskar, A. Agrawal, A. Mohan, and J. Tumblin. Dappled photography: mask enhanced cameras for heterodyned light fields and coded aperture refocusing. *ACM Trans. Graph.*, 26(3), July 2007.

[118] G. Voss, J. Behr, D. Reiners, and M. Roth. A multi-thread safe foundation for scene graphs and itsextension to clusters. In *Eurographics Workshop on Parallel Graphics and Visualization*, pages 33–37, 2002.

[119] N. J. Wade. *A Natural History of Vision*. MIT Press, 1998.

[120] G. Wallace, O. J. Anshus, P. Bi, H. Chen, Y. Chen, D. Clark, P. Cook, A. Finkelstein, T. Funkhouser, A. Gupta, M. Hibbs, K. Li, Z. Liu, R. Samanta, R. Sukthankar, and O. Troyanskaya. Tools and applications for large-scale display walls. *IEEE Comput. Graph. Appl.*, 25(4):24–33, 2005.

[121] L. Wang, Y. Zhao, K. Mueller, and A. E. Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *IEEE Visualization*, page 47, 2005.

[122] G. Wetzstein, D. Lanman, D. Gutierrez, and M. Hirsch. Computational displays: combining optical fabrication, computational processing, and perceptual tricks to build the displays of the future. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH '12, pages 4:1–4:159, New York, NY, USA, 2012. ACM.

[123] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar. Tensor Displays: Compressive Light Field Synthesis using Multilayer Displays with Directional Backlighting. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):1–11, 2012.

[124] X. Xu, S. Solanki, X. Liang, S. Xu, R. B. A. Tanjung, Y. Pan1, F. Farbiz, B. Xu, and T.-C. Chong. Computer-Generated Holography for Dynamic Display of 3D Objects with Full Parallax. *IJVR*, 8(2):33–38, June 2009.

[125] X. Zhang, C. Bajaj, and W. Blanke. Scalable isosurface visualization of massive datasets on COTS clusters. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 51–58, Piscataway, NJ, USA, 2001. IEEE Press.