**Delivering Real-Time Holographic Video**

**Content With Off-The-Shelf PC Hardware**

by

Tyeler Quentmeyer

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 5, 2004



Author _____

Tyeler S. Quentmeyer
Department of Electrical Engineering and Computer Science
May 5, 2004

Certified by _____

V. Michael Bove, Jr.
Principal Research Scientist
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by _____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

**Delivering Real-Time Holographic Video Content With Off-The-Shelf PC Hardware**

by

Tyeler S. Quentmeyer

Submitted to the
Department of Electrical Engineering and Computer Science

May 5, 2004

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

## ABSTRACT

We present a PC based system to simultaneously compute real-time holographic video content and to serve as a framebuffer to drive a holographic video display. Our system uses only 3 PCs each equipped with an nVidia Quadro FX 3000G video card. It replaces a SGI Onyx and the custom built Cheops Image Processing System that previously served as the platform driving the MIT second-generation Holovideo display. With a prototype content generation implementation, we compute holographic stereograms and update the display at a rate of roughly 2 frames per second.

Thesis Supervisor: V. Michael Bove, Jr.
Title: Principal Research Scientist

# ACKNOWLEDGEMENTS

First, I would like to acknowledge my original thesis advisor, Stephen Benton, and dedicate my work to him. His vision for holographic displays and work in holography defined the field and made all of my work possible. His work will always be an inspiration.

I am most grateful to my thesis advisor, Mike Bove. The fundamental idea behind this project was his own. He was a valuable source of information about Holovideo and Cheops. I am very thankful that he assumed supervision of my work and provided much needed leadership. Without his guidance and support, I would not have been able to achieve my goals.

I would like to thank everyone in the Spatial Imaging group, past and present, particularly Wendy Plesniak, Steve Smith, Pierre St.-Hilaire, and Sam Hill. Wendy was a wonderful source of information about Holovideo and an invaluable collaborator in hologram computation. I would like to thank Steve for helping get this project off the ground, for his guidance and input, for assuming leadership of the group, and for helping take photographs and make movies. I would like to thank Pierre for building the display on which my work was based and for his input about working with the system. I would like to thank Sam for his input, for helping take photographs, and for helping me maintain my sanity.

Thanks also to Won Chun and Joe Duncan. Won was a great source of information about OpenGL, optimizing my rendering algorithms, and 3D software in general. Joe provided helpful information about some of the electrical underpinnings of the system.

Finally, I would like to thank my parents, Steve and Becky Quentmeyer, for their lifelong encouragement. Without their support and guidance, I would not have made it this far.

# TABLE OF CONTENTS

# 1 INTRODUCTION

## 1.1 Motivation for improving 3D display technology

The proliferation of computing devices and increasingly complex electronic data in our daily lives is driving the need for effective visualization tools. The importance of understanding and manipulating three-dimensional data sets goes without saying in a number of fields such as computer-aided design, engineering, medical imaging, navigation, and scientific research. Possible applications for a three-dimensional visual experience to the arts and entertainment industry are endless. The motivation for improving display and content creation systems is overwhelming.

Although CPU speeds have been increasing exponentially with Moore's law, display technology has stagnated. In fact, it has been the slowest improving technology in the computing industry. Relative to CPU speeds, display technology has changed very little since the introduction of the television tube. CRT monitors have given way to LCDs and screen resolutions have seen improvements, but the fundamental experience has changed little since the advent of computing.

The typical 2D monitor used to interface with digital data sets only takes advantage of a small fraction of the amount of information that can be processed by the human visual system. Not only do 2D displays suffer from low resolutions, they do not take advantage

of stereo vision. Humans rely on depth cues to understand 3D data that are not provided by 2D displays.

Understanding 3D data sets is an important task. To put the mission in perspective, consider a few applications in medical imaging. MRI scans have the ability to collect enormous amounts of data about a patient. The data is usually presented in the form of thousands of 2D slices, making it very difficult for doctors to first find areas of interest and then to develop a coherent 3D understanding of the information at hand. MRI data is relied on for a number of tasks including identifying brain lesions, tumors, and internal trauma and for planning surgical procedures. Additionally, MRI capture technology is improving much faster than our ability to display it. This means that although we can get better pictures of the innards of the human body that in principle lead to better health care, we are limited by the ability of doctors to interact with and understand the massive amounts of data gathered by MRI technology.

Consider another compelling application of 3D imaging proposed by Plesniak [16]. If we couple a 3D display with a force feedback device, we not only allow the user to visualize a 3D data set but also to feel and natively manipulate that data set. By coupling a haptic device with three-dimensional display technology, we can build a system where a doctor can practice surgical procedures using a force feedback scalpel on a computer simulated holographic patient.

## 1.2 **The MIT second-generation Holovideo system**

The MIT second-generation Holovideo system is a real-time electro-holographic display. The end result of the system is a single-color horizontal parallax only (HPO) holographic image that is updated at video speeds. The three-dimensional image fills a volume 150mm wide, 75mm high, and 160mm deep, is visible over a range of 30 degrees, and is refreshed 30 times per second [1]. The system is made up of three components: the display (Holovideo), a framebuffer with stream-processing capabilities to serve holograms to the display (Cheops), and a computation platform to compute holographic data to be shown on the display (SGI Onyx).

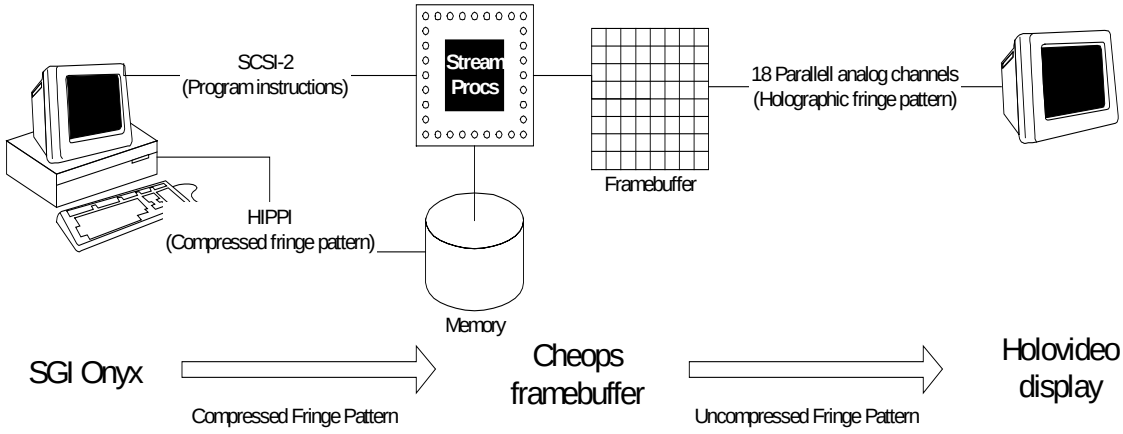



**Figure 1: Overview of the MIT second-generation Holovideo system.**

Holovideo is a display that inputs a 36MB computed holographic fringe pattern and outputs a reconstructed holographic image 30 times per second. The display reads its input over a custom 18-channel parallel connection. We will discuss Holovideo in more detail in Chapter 2. [1]

The Cheops Image Processing System is a framebuffer with special purpose embedded stream-processing capabilities. It reads program instructions for its processors over a SCSI-2 bus and reads fringe pattern data directly into memory over a HIPPI bus. Cheops then runs the uploaded program on its stream-processors and fills its framebuffer memory with a holographic fringe pattern. The framebuffer is connected to Holovideo via the custom 18-channel parallel connection. Cheops is discussed in more detail in Chapter 4. [5][21]

The SGI Onyx is a general-purpose computing platform used to compute compressed holographic content that is loaded into Cheops and displayed by Holovideo. The SGI is not used to directly feed Holovideo for two reasons. First, Holovideo's high bandwidth requirements (36MB/frame * 30 frames/second) make it impossible. Second, the decompression algorithm run by Cheops is in fact a post-processing step that would have to be run on the SGI anyways. Cheops stream-processors not only alleviate the SGI of this extra step, but perform it faster than the SGI could. Each compressed fringe pattern is uploaded to Cheops, uncompressed, sent to Holovideo, and finally displayed as a three-dimensional image. [2][5]

## 1.3 **Motivation for PC platform to drive Holovideo**

There is a substantial amount of motivation to remove the SGI Onyx and the Cheops framebuffer from the Holovideo system. The Onyx is an outdated machine that is difficult to upgrade and difficult to replace without affecting the system. It has been prone to failure and requires a $6,000 annual service contract ($15,000 before a $9,000

educational discount) to keep it up and running. The Cheops system has proven very reliable but was custom designed and built. It is therefore virtually impossible to upgrade without rebuilding the entire system. In the event of its eventual failure, it is also extremely difficult to replace.

Off the shelf PCs and video cards seem to be an ideal replacement for the SGI and Cheops. PCs can replace the Onyx and Cheops to both compute holograms and to serve as a framebuffer for Holovideo. Modern video cards are incorporating a rendering pipeline that is sophisticated enough to implement certain holographic rendering algorithms. The high demand for and the high volume in which they are produced ensure that PC components are cheap, reliable, and easy to replace. They can be swapped out and upgraded seamlessly when new technology becomes available. PC CPU speeds improve with Moore's law and PC video card speeds improve at three times Moore's law. In this way, the system driving Holovideo can improve in speed, reliability, and cost effectiveness at the same rate as the market for off the shelf PC components.

## 1.4 **Outline of thesis**

The goal of this thesis is to build a platform to serve as a framebuffer for holographic video display systems, particularly the MIT second-generation Holovideo system, that is also capable of computing holographic content in as close to real-time as possible. We want to completely remove dependence on the SGI Onyx and Cheops Image Processing System to run Holovideo. Using only commodity PC hardware that is reliable, easy to replace, and relatively inexpensive, we want to construct a system that is capable of

serving as a framebuffer for Holovideo and is capable of computing holograms for Holovideo and writing them to the framebuffer at rates as close to smooth video frame rates as possible.

The next three chapters outline and give details about the systems we need to understand in order to achieve our goals. Chapter 2 discusses the Holovideo display at the level in which we need to understand it – mostly in terms of its inputs and outputs. Chapter 3 discusses how holograms are constructed, beginning with a very brief introduction to optical holography and then introducing computed holograms with emphasis on diffraction specific hologram. Chapter 4 describes the architecture and capabilities of the Cheops system that we are trying to replace.

The following two chapters introduce the bulk of the work for this thesis. Chapter 5 gives an overview of the PC architecture we introduce to drive Holovideo. It then gives details about how our PC system serves as a framebuffer for Holovideo. Chapter 6 discusses how our PC system can be used to compute holograms. It includes a discussion of the computational capabilities of our system and the description of a prototype implementation to compute diffraction specific holograms in real-time.

The final three chapters conclude this thesis document. Chapter 7 gives the results of our project, including a discussion of the image quality we produce and of the computation speeds we were able to achieve. Chapter 8 gives a few directions for future work and Chapter 9 offers a few concluding remarks.

# 2 HOLOVIDEO SPECIFICATIONS

## 2.1 Introduction

In this chapter, we give a brief overview of the MIT second-generation Holovideo display and a detailed specification for its inputs. For complete details, see Pierre St. Hilaire's doctoral dissertation [1]. Note that although our PC hologram computation and delivery platform is not specific to this holographic video display, future displays are likely to require similar inputs so we use Holovideo as a concrete example and proof of concept.

## 2.2 Holovideo Overview

A holographic display uses a computed fringe pattern to modulate light and produce a three-dimensional image. The crux of holographic video is the spatial light modulator (SLM), a device that modulates light with a computed fringe. Holovideo uses two cross-fired 18-channel acousto-optic modulators (AOMs) as the SLM and a chain of optics and scanning mirrors to construct a horizontal parallax only (HPO) hologram at video frame rates. The output of Holovideo is 144 vertically stacked horizontal lines, each of which is a thin HPO hologram (called a hololine), which are updated in real-time.

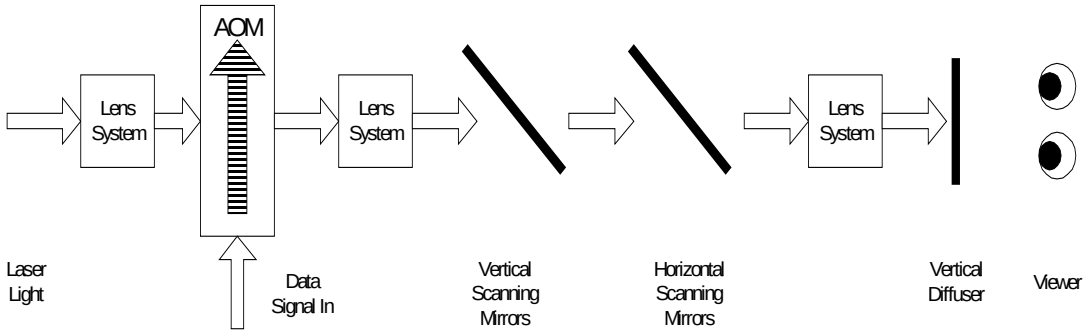



**Figure 2: Overview of the Holovideo display architecture.**

Fringe patterns for the hololines in analog format are read from some storage unit (Cheops in the old system and the PC framebuffer in the new system) in groups of 18 and passed to Holovideo's 18 data input channels. Each fringe is then passed to a radio frequency (RF) process unit that frequency shifts the fringe to the AOM's desired frequency range. From there, the fringe is input in to one of the AOM's 18 input channels. Each output from the AOM is then passed to a system of scanning mirrors that steers the modulated light to the correct horizontal and vertical position. Finally, the diffracted light is imaged on a vertical diffuser at the output of the display. In this way, 18 hololines are imaged in one step. The process is repeated 8 times, until all 144 hololines are imaged.

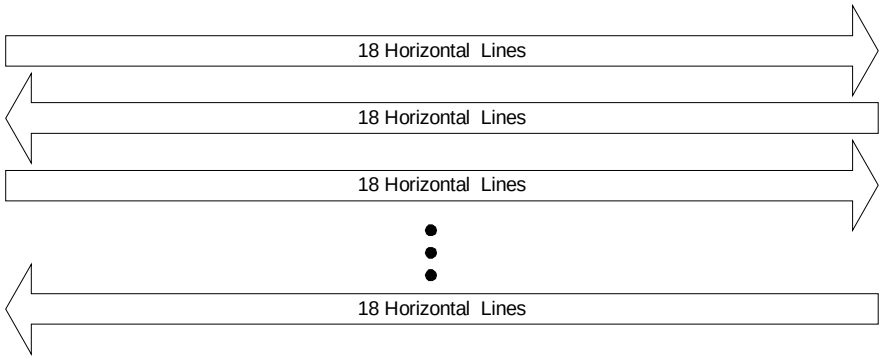



**Figure 3: The boustrophedonic scanning pattern used by Holovideo to draw lines to the image plane.**

The system of mirrors that steers the modulated light consists of a vertical scanning system and a horizontal scanning system. On input of the first set of 18 fringe patterns, the horizontal scanning system steers the fringe pattern from left to right over the horizontal range of the image. On the next set of inputs, the horizontal scanning system steers the fringe pattern in the opposite direction, from right to left. This boustrophedonic pattern removes the need for a horizontal retrace and thereby eliminates wasted time between horizontal scans. However, it also means that every other fringe pattern is imaged backwards and therefore needs to be fed into Holovideo in reverse order. The vertical scanning system lays down fringe patterns from top to bottom for each frame.

Between frames, the vertical scanning mirror needs to return to its starting position. In order to allow it to do so, there is a vertical retrace time between frames equal to one complete horizontal scan (left to right and back to left). Between horizontal lines, the horizontal scanning mirrors need to slow to a stop and accelerate to their scanning velocity in the opposite direction. While the horizontal mirrors are imaging lines, they need to move at a constant velocity to avoid distorting the image data. The horizontal mirrors therefore cannot be used to image data while they are nonlinearly changing directions. To compensate for this, there is a horizontal blanking period between fringe patterns on each data line of roughly 0.9ms. This value was determined empirically. For the display's scanning geometry, each horizontal line is scanned in a total of 3.3ms, giving a blanking period of about 27.27 percent.

## 2.3 **Output characteristics**

2.3.1.1 Image size and view zone

Holovideo's holographic output images into a view zone 150mm wide, 75mm high, and 160mm deep. (The depth of the view zone is in principle limited only by the amount of astigmatism that the human eye can tolerate at a typical viewing distance, 300mm. However, in practice, the 160mm depth figure is accurate.) The horizontal viewing angle – the angle from which the viewer can see the image – is 30 degrees.

### 2.3.2 **Resolution**

Each fringe pattern is $2^{18}$ (256K) samples in length laid down over the 150mm wide image zone, giving a horizontal resolution of 256K/150mm = 1,748 samples per millimeter. The horizontal resolution is high enough to diffract light without artifacts visible to the human eye. There are 144 vertical lines over the 75mm high image zone, giving 2 lines per millimeter, equivalent to a 19" NTSC display. The value of 256K samples was chosen because it is easy to provide with the Cheops framebuffer and because it provides a data frequency suitable to the display characteristics.

## 2.4 **Inputs**

To understand the inputs to Holovideo, imagine that the fringe patterns are being stored in 18 parallel video framebuffers. Each fringe pattern is on one horizontal line. Therefore each framebuffer provides 8 horizontal lines per vertical refresh.

### 2.4.1 **Horizontal sync signal**

Holovideo reads in a horizontal sync signal from our imaginary framebuffer. A rising edge should coincide with the start of a new fringe pattern on the data inputs (a phase delay between the fringe pattern location and the horizontal sync pulse is set on a signal generator as described in 2.4.4.2). The width of the pulse is ignored – only the rising edge is used.

### 2.4.2 **Vertical sync signal**

Holovideo also reads in a vertical sync signal from our imaginary framebuffer. For a particular frame, a rising edge should coincide with the end of the last fringe pattern's horizontal blanking period and therefore the beginning of the vertical retrace period. Although the width of the pulse is ignored, it must be of size less than one horizontal sync period. Following the rising edge of the vertical sync signal, the next two horizontal sync periods should not contain fringe patterns.

### 2.4.3 **Data inputs**

Holovideo has 18 data input channels. Each input channel reads an analog signal corresponding to fringe patterns at a frequency dictated by the horizontal sync signal. Each fringe pattern should consist of 256K samples followed by a blank period determined by the system's horizontal blanking period. Each data channel should be driven with a series of 8 fringe patterns, followed by 2 blank fringe patterns. The 8 used fringe patterns should alternate being in the left-to-right and right-to-left formats,

beginning with left-to-right. That is, the first fringe should correspond to an image from left to right, the second fringe should correspond to an image from right to left, et cetera.

2.4.3.1 Hologram data format

The time multiplexed fringe patterns from the 18 data input channels are combined by Holovideo to create a single frame of an image as follows: The first fringe from the first input is mapped to the first horizontal line of the image output, the first fringe from the second input is mapped to the second line, the first fringe from the third input to the third line, et cetera. Then, the second fringe from the first input is mapped to the 19$^{th}$ line, the second fringe from the second input to the 20$^{th}$ line, the second fringe from the third input to the 21$^{st}$ line, and so on.

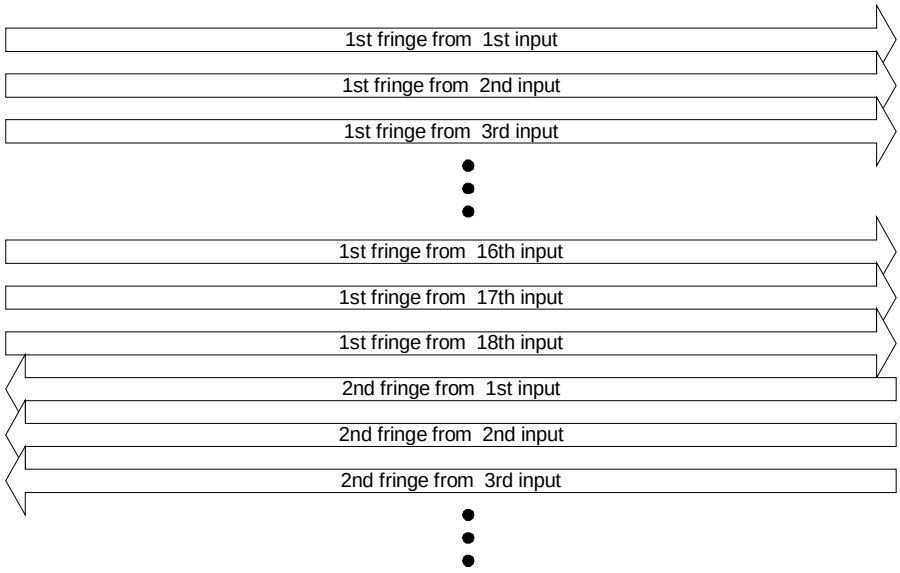



**Figure 4: The mapping from fringe patterns on the 18 parallel inputs to the lines drawn on the screen.**

### 2.4.4 **Horizontal scanning signal generator**

The horizontal scanning system of Holovideo is driven with a signal generator that produces a triangle wave. The beginning of the triangle wave coincides with the

beginning of the left-to-right scan, the peak coincides with the end of the left-to-right scan and the beginning of the right-to-left scan, and the end coincides with the end of the right-to-left scan. The start of a triangle wave period is triggered by the horizontal sync signal input. Strictly speaking, the horizontal scanning signal generator is part of the scanning system and not an input. However, the triangle wave generated by the signal generator has properties that must be set by hand to match the data input system.

#### 2.4.4.1 Frequency

The frequency of the triangle wave generated by the signal generator should be twice the frequency of horizontal sync pulses.

#### 2.4.4.2 Phase

The phase of the triangle wave generated by the signal generator should be set to match the offset of the rising edge of the horizontal sync pulse and the start of a fringe pattern on the data channels.

# 3 COMPUTING HOLOGRAMS

## 3.1 Introduction

In this chapter, we give a cursory background on traditional optically generated holograms and on synthetically generated computed holograms. We begin with an overview of the simplest optically generated hologram. We then introduce computationally generated holograms through interference modeling. Finally, we introduce Lucente's diffraction specific holographic stereogram and the Hogel-Vector compressed encoding thereof.

## 3.2 Optically generated holograms

The simplest hologram to describe is the in-line (Gabor) hologram. This method is limited to creating a hologram of a translucent image located on a transparency, yet it serves as a useful example. A transparency prepared with a translucent image to be turned into a hologram is illuminated head-on with a collimated light source. The diffracted light pattern is then recorded on a photographic plate located opposite the transparency. When the photographic plate is later illuminated with the same light source used in the recording process, an image of the transparency appears to float in space. [7]

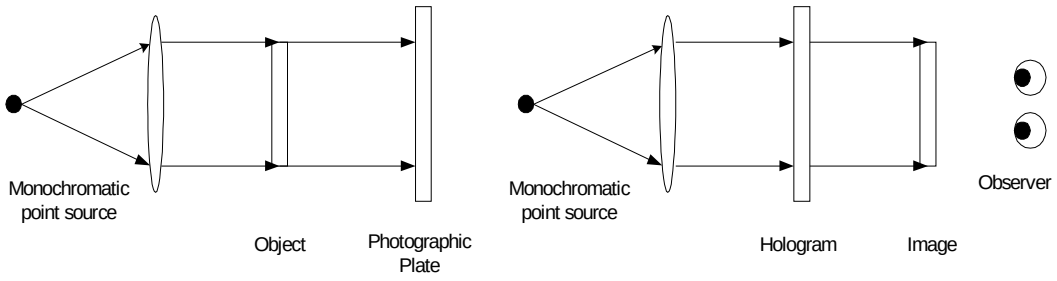



**Figure 5: The in-line hologram setup. The first image is the setup used to create the hologram and the second is the setup used to view the hologram.**

The photographic plate records the intensity of the impending electric field at each point. There are two parts to this field: one from the light source and one from the light interacting with the object. The collimated monochromatic light source provides a simple plane wave traveling orthogonally towards the transparency.

$$E_{source} = E_{plane} = e^{i(kr-wt)}$$

Light from the source reflects off the object and provides some complicated field depending on the shape and surface properties of the object.

$$E_{object} = f(r)$$

At the photographic plate, the electric field is the superposition of the two fields and the intensity is the modulus squared. The intensity at the plate is called the diffraction pattern or the fringe pattern of the object.

$$I_{hologram} = |E_{source} + E_{object}|^2$$

Later, when the photographic plate is again lit with the source, the light interacts with the recorded fringe pattern to approximately reconstruct the electric field that would be present if the object itself was being illuminated by the light source.

There are two major differences between the in-line hologram and more general methods. First, the light source plane wave (typically called the reference beam) is usually tipped at some angle $\theta$ with respect to the photographic plate. Second, the object to be imaged is usually three-dimensional rather than a flat transparency. The hologram is, however, still just the diffraction pattern from a plane wave and some object.

## 3.3 **Interference holograms**

Although the simple setup described above is far from state of the art, optically generated holograms will always be limited to static representations of objects that actually exist and are practical to manipulate in a controlled laboratory setting. We would have a very hard time, for example, making a traditional hologram of the Eiffel tower or a holographic movie of a ball bouncing.

Since a hologram is just a photographically recorded diffraction pattern, it is easy to model the process computationally and print a computer generated diffraction pattern. Rather than simulate the light source interacting with the object, we can model the object as a densely packed skin of analytically defined light emitters and simply compute the superposition of the reference beam and each of the object's emitters. The simplest type of emitter is the spherical emitter, which outputs light uniformly in all directions.

$$E_{spherical} = Ae^{i(k|r-r_0|-wt)}$$

The field at the photographic plate is then just

$$E_{total} = \sum E_{spherical} + E_{reference} = E_{object} + E_{reference}$$

where $E_{object}$ is the sum of the emitter fields (the field from the object). This gives an intensity of

$$I = |E_{total}|^2 = |E_{object}|^2 + |E_{reference}|^2 + 2Re\{E_{object} * E_{reference}\}$$

This intensity value is calculated over the entire discretized photographic plate and output using the equivalent of a very high quality printer and then lit in the same way as a normal hologram.

This approach to computing holograms was pioneered and successfully demonstrated by Leseberg in 1986 [9]. Higher quality images using the same basic principles were demonstrated by Underkoffler [22]. Holograms computed using the interference method are some of the highest quality computer generated holograms made to date. They are, unfortunately, also very slow to compute.

Interference computed holograms are high quality and in a sense the best holograms that a digital system can generate. They are, however, slow to compute. For an object consisting of 10,000 spherical emitters and our 36 megasample display, interference computation will take at least 5 trillion floating-point operations. To compute the thirty holograms per second necessary to generate dynamic content, we would need a computing system capable of 60 trillion flops – far from feasible with existing hardware. [2]

## 3.4 **Diffraction specific holograms**

The inspiration for diffraction-specific computation of holograms comes from examining the functional role of the fringe pattern. When a light source impinges on the fringe pattern, the light from the source is diffracted in some direction. The job of the fringe pattern, then, is to diffract light in a particular direction. As it turns out, the angle at which light is diffracted is a simple function of the spatial frequency of the fringes. If we want to diffract light at some angle $\theta$, all we have to do is output a fringe pattern with spatial frequency $f(\theta)$. A fringe pattern that does this is called a basis fringe. The intensity of the light diffracted by a basis fringe can be manipulated by scaling the amplitude of the fringe. [2] [19]

We want to fill the view volume with light in every direction, so the fringe pattern is obviously more than a single basis fringe. Since we are constructing a three-dimensional image, the light you see when you look at the display from one direction should be different from the light you see in another direction (i.e., the object should exhibit parallax as you move your head from left to right). Let the intensity of light you see from angle $\theta$ be called $W_\theta$. We want to construct a fringe that diffracts $W_\theta$ light into angle $\theta$ for all $\theta$. The fringe to do this is

$$fringe = \sum_{\theta} W_\theta * basis_\theta$$

If we were to fill each hololine with one fringe pattern computed in this way, the image would always be a single color at each angle. We want to divide the hologram into

subregions of some vertical and horizontal extent. In our case, we use vertical stripes. This way, at any given angle, each vertical stripe contributes a single color for some discrete horizontal region. One of these bars for a single horizontal line is called a holographic element, or hogel for short. The composite image is then much like a normal digital image built-up from a number of uniformly colored pixels. A typical configuration for the Holovideo display is to divide each horizontal line into hogels 1,024 samples long. Since a horizontal line is made up of 262,144 samples, there are 256 hogels per hololine.

The equation describing a fringe pattern as the weighted sum of basis fringes given above is in terms of a continuous viewing angle parameter, $\theta$. The Holovideo display is digital so we must discretize $\theta$. After examining the optical properties of the display and the human visual system, a typical value of 32 discrete viewing angles over a range of thirty degrees is used. This means that the range of angles at which the display can be viewed is thirty degrees (fifteen degrees to the left or right). In that arc, 32 distinct images are displayed at evenly spaced angular increments. Since there are 32 angular regions, we need 32 basis fringes. Each basis fringe is now responsible for diffracting light over a small angular increment so is redefined to contain all spatial frequencies that map to angles within its increment.

The algorithm for computing a diffraction-specific hologram is now easy to outline. First, the 32 basis fringes are computed. Then, 32 images, each 256 by 144 pixels, are rendered from evenly spaced positions along a horizontal line at eye level using any

standard computer graphics method. For each horizontal line, each hogel is then computed using the equation given above for a fringe in terms of weights and basis fringes where each weight is the corresponding pixel value from one of the 32 rendered images. The complete horizontal line is 256 sequential hogels and the complete hologram is 144 stacked lines. By using a discrete set of images and view zones to approximate a continuous optical phenomenon, we are constructing what is called a holographic stereogram [17].

### 3.4.1 **Hogel-Vector encoding**

The current Holovideo display is configured with a HIPPI bus connecting the Cheops framebuffer and a SGI Onyx workstation. The amount of data that can be transferred from the Onyx to Cheops is therefore limited by the bandwidth of the HIPPI bus. Sending a fully computed fringe pattern of 36 MB 30 times per second would require over 1GB/s of bandwidth. Since the HIPPI bus runs at 100MB/s, we need a way of compressing a fringe pattern by at least a factor of ten in order to achieve video frame rates. Cheops must be capable of constructing the full fringe pattern from the compressed form. [2]

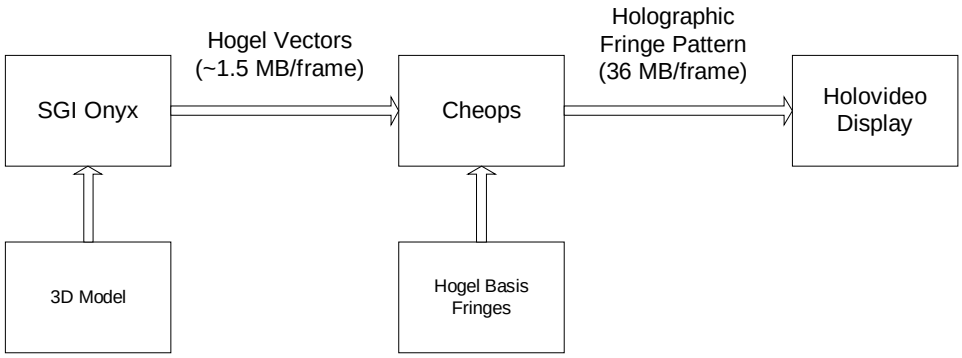



**Figure 6: The flow of hologram data using Hogel-Vector encoding.**

Rather than compute the fringe pattern at each hogel, we can express all of the required information in terms of the basis fringes and the weights. We organize the corresponding weights from each rendered image in each hogel into a vector called a Hogel-Vector. For example, the Hogel-Vector for the 17$^{th}$ hogel contains the color value from the 17$^{th}$ pixel of each of the 32 rendered views. We organize the basis fringes into a set of vectors with one vector for each sample in a basis fringe (for a total of 1,024 vectors). The basis fringe vector for a sample point contains the sample value from each of the 32 basis fringes corresponding that sample point. For example, the 567$^{th}$ basis fringe vector contains the 567$^{th}$ sample of each of the 32 basis fringes. The i*th* sample of a fringe pattern is then calculated as

$$fringe_i = hogel\ vector * basis\ fringe\ vector_i$$

Since the basis fringes never change, we can send them to Cheops at system initialization. For each holographic frame, then, we only need to send the Hogel-Vectors. Each Hogel-Vector is a 32 element vector with 8 bits per element. A full hologram is 256 hogels wide and 144 tall for a total of 36,864 hogels or 36MB of data. This gives a compression ratio of about 1,000, good enough to send all of the required information to Cheops in

real time. The compression format is lossless and simple enough that the Splotch Engines on Cheops are capable of constructing the uncompressed holograms from the Hogel-Vector encoded format.

# 4 CHEOPS

## 4.1 System Overview

The Cheops Image Processing System is a block data-flow parallel processor originally designed for research into scalable digital television at the MIT Media Lab. A custom Cheops configuration was built primarily to serve as a framebuffer to feed holographic fringe patterns to Holovideo. Its secondary purpose is to provide high-speed custom computational power to aid in computing fringe patterns. [5][21]

Cheops is a modular system that connects input modules, output modules, and computational blocks. These modules are connected through two busses: The first, the Global Bus, is a 32-bit wide bus designed for transferring control instructions at rates up to 32 MB/s. The second, the Nile Bus, is designed for transferring large blocks of 24-bit wide data at sustained rates of 120 MB/s.

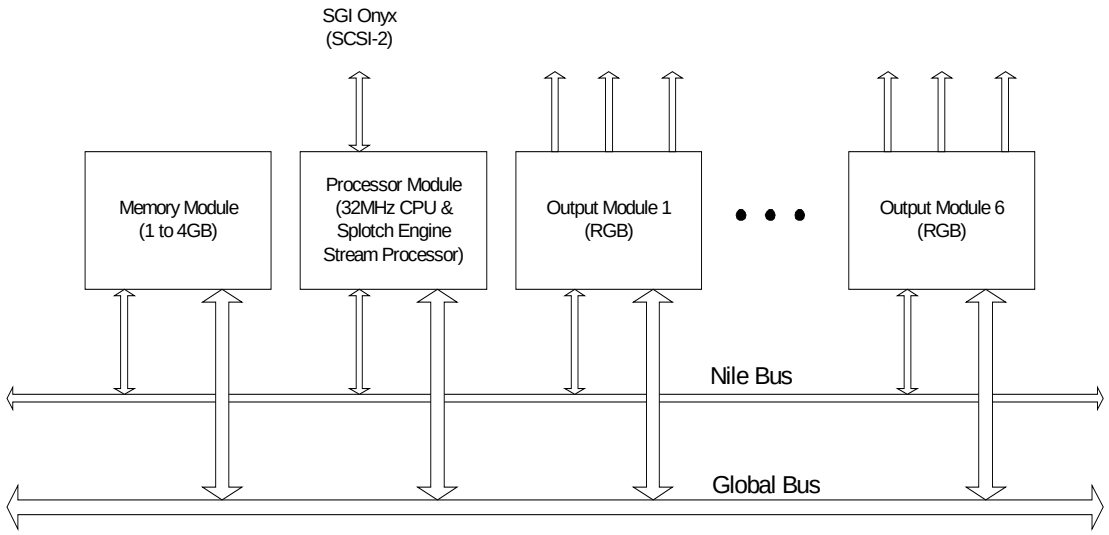



**Figure 7: The Cheops configuration used to drive Holovideo.**

The configuration used for the MIT second-generation Holovideo system has 6 output modules, each with 3 output channels. It has a memory module that provides 1 to 4 GB of local storage and a HIPPI input module that reads and writes data to the memory module at 100MB/s. Finally, the configuration has a processor module configured with a custom stream-processing daughter card called the Splotch Engine. Each processor module has a SCSI-2 input over which it reads instruction sequences.

## 4.2 **Processor module**

The Cheops processor module is a generic processing unit that accepts custom daughter cards to perform specialized tasks. The idea behind the processor module is to decouple computationally intensive tasks from the framebuffer system. A custom daughter board with specialized hardware performs a specific task over a high-throughput memory interface under the control of a general-purpose processor that resides on the processor module. Since image processing applications access data in a regular fashion, rather than

requiring that each custom daughter board provide a large local memory or that it be able to randomly access the Cheops main memory at high speeds, the custom daughter cards operate on one or more high-speed streams of data.

Each processor module has eight memory units that communicate via a crosspoint switch with up to eight stream processing units. Each memory unit can transfer a stream of data through the switch at up to 32 MSamples/s (for 16 bit samples) and can store 4 MB of data. Up to four processing pipelines (a stream source, a stream processor, and a stream destination) can occur simultaneously. Each port for a stream processor can input and output up to two data streams. If a stream processor requires more than two input and two output streams, it can use multiple ports.

A general purpose 32 MHz 32-bit CPU (an Intel 80960CF) on the processor module is provided to initialize and control the flow of data between the different functional units, to implement algorithms that are not available in stream processing units, and to communicate with the outside world.

In addition to connections to Cheops' Global Bus and Nile Bus, each processor module is equipped with a SCSI-2 bus over which it communicates with a computer. The computer sees the processor module as a fixed disk device. The SCSI-2 bus is used to load application code and data sets from the computer to the processor module at speeds of up to 1.5 MB/s.

## 4.3 **Output modules**

The primary purpose of the output modules is to decouple the difference in speeds between data output and data transfer and computation. For example, the output modules allow Cheops to maintain a 30 frames/s refresh rate even when holographic fringe pattern computation occurs at a more moderate 0.1 to 3 frames/s.

The Cheops configuration used for Holovideo has 6 output modules. Each module contains three 8-bit data output channels (normally the red, green, and blue channels of a full color framebuffer), a horizontal sync channel, and a vertical sync channel that output a configurable analog video signal. The data for each channel is read from a 2 MB memory bank, for a total of 18 output channels read from a parallel 36 MB memory bank. These channels are connected to Holovideo's 18 inputs, serving as a framebuffer to drive the display. Refer to the video mode section for more details about the horizontal and vertical sync signals.

Each output module in the Holovideo configuration is a standard Cheops output module with slight modifications. The modules were modified to synchronize their output scanning and sample clocks. This means the data on each of the 18 channels is synchronized, a feature known as genlock. When the first module is reading and outputting the first sample from its memory bank, so are the rest of the modules.

### 4.3.1 **Framebuffer specifications**

Each output channel outputs a video mode that is 256K (262,142) samples wide and 8 lines tall at 30 Hz. There is a horizontal blanking period between horizontal lines of 0.9 ms, equal to about 93,304 samples. Between frames, there is a vertical sync pulse one line in length followed by a vertical blanking period also one line in length. The horizontal sync pulse is output on the horizontal sync channel and uses positive polarity. The vertical sync pulse is output on the vertical sync channel and uses positive polarity.

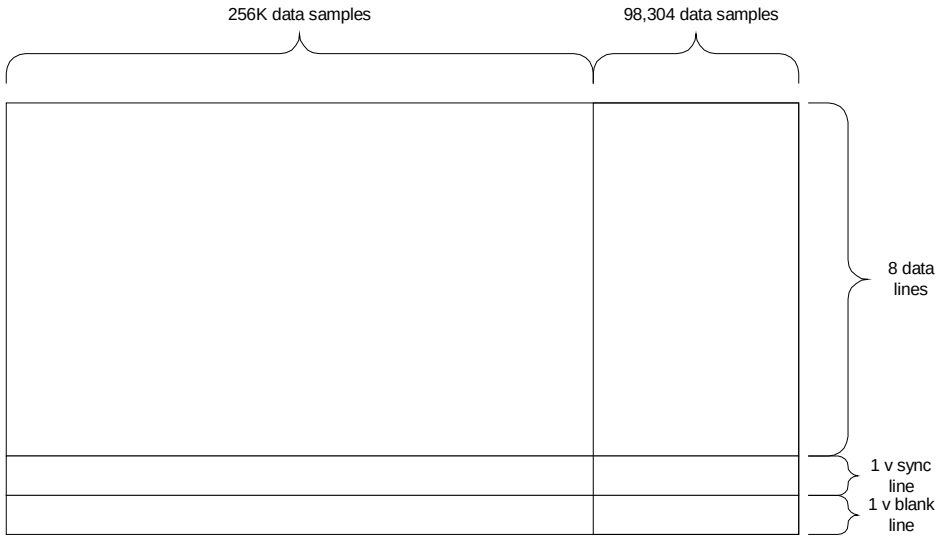



**Figure 8: Video mode output by each Cheops output module to drive Holovideo.**

## 4.4 **Splotch Engine**

The Splotch Engine is a custom module that can be placed on a processor module to perform modulation and accumulation [5]. It was designed to perform the Hogel-Vector decoding step to compute diffraction specific holograms. Each Splotch Engine is capable of performing 4 modulation and accumulation steps in parallel at the rate of one per clock cycle with a maximum of a 40MHz clock. Our system uses a 32MHz clock.

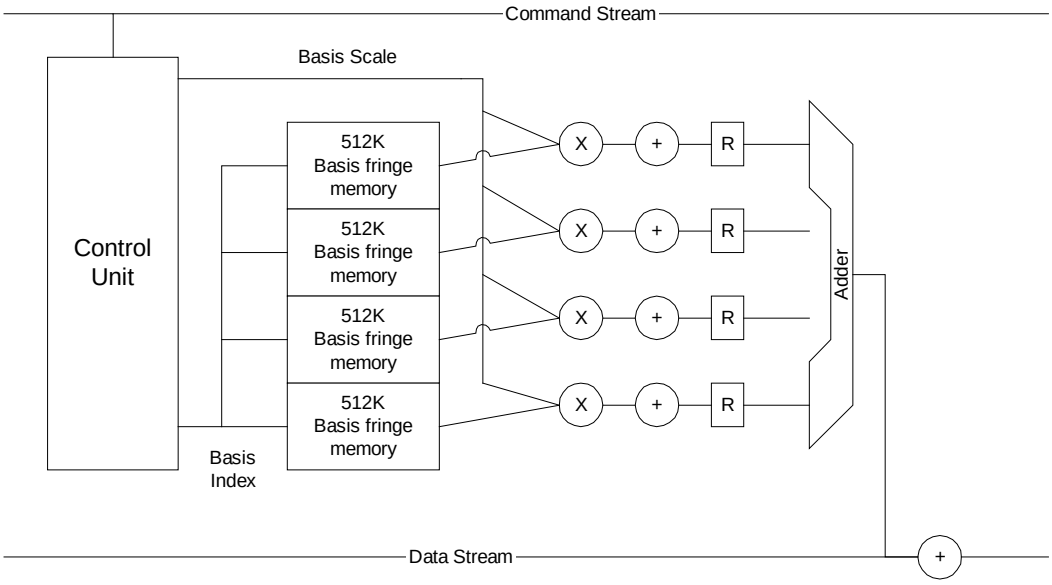



**Figure 9: Splotch Engine block diagram.**

The module has a 512K memory per modulation and accumulation element in which the basis fringes are stored. It takes two input streams and produces two output streams: a command stream and an input stream. The command stream contains a set of commands for each of the four modulation and accumulation units. Each command set gives the modulation unit a weight and tells the modulation unit which basis fringe to read from and which sample value from that basis fringe should be multiplied by the given weight value. The outputs of the four modulation units are then summed together and accumulated with the data input stream to yield to the output data stream value.

### 4.4.1 **Hologram computation speeds**

Each Splotch engine is capable of modulating and accumulating 4 out of the total 32 entries in a Hogel-vector. Each processor module can be configured with up to 3 Splotch Engines. Each processor module is therefore capable of modulating and accumulating 12 out of 32 entries in a Hogel-vector in parallel. Fully decoding a Hogel-vector requires a

minimum of 3 passes through the parallel pipeline. In our Cheops configuration with two Splotch Engines, computing a hologram for Holovideo took two seconds, for a frame rate of 0.5 frames per second.

# 5 USING PCS TO DRIVE HOLOVIDEO

## 5.1 Introduction

The goal of this project was to build a framebuffer for Holovideo from off the shelf PC hardware. The obvious choice for generating the input signals is a collection of video cards. We need to map the outputs of video cards to the three inputs of Holovideo: the 18 parallel data inputs, the horizontal sync input, and the vertical sync input. In this chapter, we first give a short overview of our PC based architecture. We then go through each of the inputs to the Holovideo display, analyze the requirements for the input, and show how our system provides a signal that meets those requirements.

## 5.2 System overview

The new platform to drive Holovideo consists of three PCs, each with a single nVidia Quadro FX 3000G. Each Quadro FX 3000G is configured in dual-head mode and therefore outputs two sets of red, green, and blue, for a total of six output channels from each card. The six outputs from each of the three cards are synchronized using the Quadro FX 3000G's frame lock feature. The 18 outputs are sent directly to Holovideo's 18 parallel data inputs. Holovideo's vertical sync input is driven directly by the vertical sync signal from one of the video cards. The horizontal sync output from one of the cards is converted from the PC video mode to Holovideo's video mode by a dedicated TTL circuit. The output of the converter circuit is used to drive Holovideo's horizontal sync input.

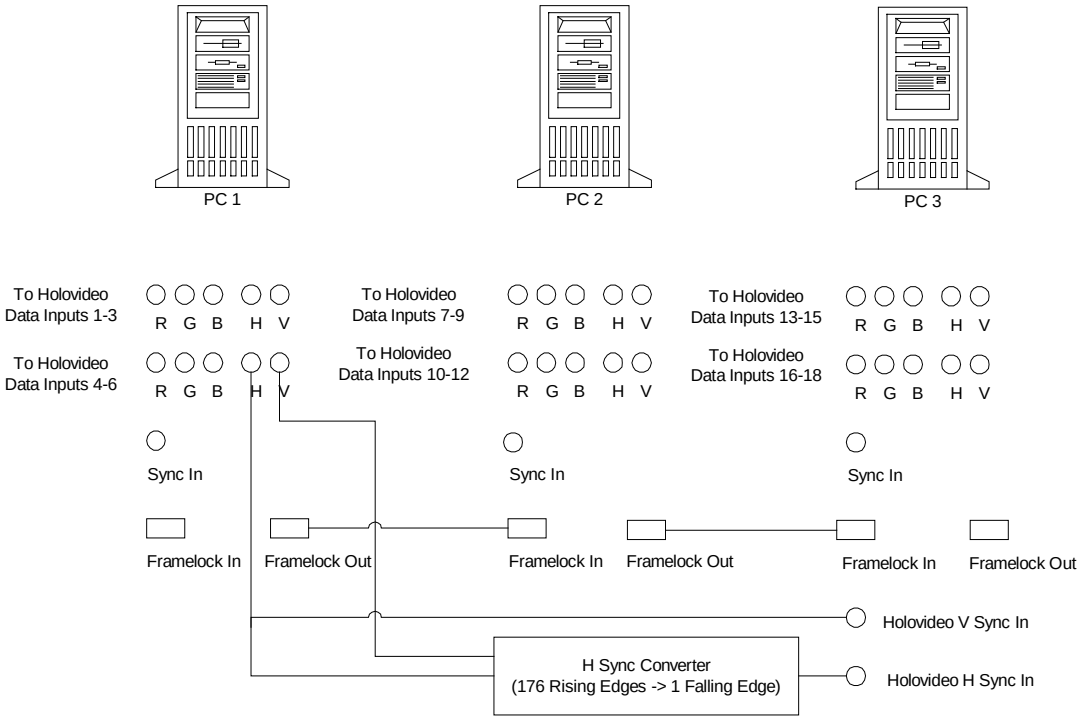



**Figure 10: Architecture overview of the new system.**

## 5.3 **Data inputs**

### 5.3.1 **Requirements from Holovideo**

To supply the data inputs, the new system must be capable of outputting 18 synchronized channels. Each channel must be capable of outputting 8 sequential blocks of 256K samples (for a total of 2,097,152 data samples) followed by 2 sequential blocks of 256K zero-valued samples. Blocks must be spaced by the 0.9ms horizontal blanking time of the display. The sequence of 10 blocks of 256K samples must be repeated at 30Hz.

### 5.3.2 **Choice of video card**

A standard video card that drives a single display outputs three synchronized signals (one for red, one for green, and one for blue), each suitable for driving a data input channel. In order to build up the necessary 18 synchronized parallel data inputs, we need video cards

that synchronize their output to a common source. The technology that allows a video card to accept a timing input and therefore to synchronize its output with an external timing source is called genlock.

Although many modern video chips have the ability to synchronize to an external source (including those from ATI, nVidia, and 3Dlabs), it is a rarely used feature on PCs and is therefore not exposed by most video card manufacturers. At the time of this writing, there are only two mass produced commercially available chips that support the genlock feature: the nVidia Quadro FX 3000G and the 3Dlabs Wildcat II 5110-G. For driver quality, hardware performance, and future upgradability, we chose to use the Quadro FX 3000G. PNY is currently the only board manufacturer that makes a video card based on the Quadro FX 3000G.

### 5.3.3 **nVidia Quadro FX 3000G output specifications**

The Quadro FX 3000G has support for driving two displays. Each output is equipped with a 400MHz DAC [11]. The two outputs are driven by the same chip with the same timing source; therefore, the two sets of RGB outputs on a single card are synchronized.

#### 5.3.3.1 Synchronizing outputs

There are two features that enable multiple Quadro FX 3000Gs to synchronize their output signals: genlock (also known as frame sync) and frame lock. [12]

##### 5.3.3.1.1 Genlock

The video card accepts a BNC genlock input to which it can match its video mode timing with several configurable parameters. The genlock input can accept a NTSC/PAL, HDTV, or TTL format timing source. The drivers support synchronizing to the genlock input with a configurable input polarity and phase delay from the timing trigger. They also support sampling the input timing source by ignoring a configurable integer number of input triggers.

5.3.3.1.2 Frame lock

Frame lock allows the video card to synchronize output frames across multiple Quadro FX 3000Gs. The frame lock input allows a group of video cards to synchronize both frame redraws (synchronize vertical sync pulses) and frame buffer swaps (synchronize changes to the output data). The video card accepts a RJ45 frame lock input and provides a RJ45 frame lock output. In this way, video cards can be connected with a linear chain of Ethernet cables to synchronize their output channels.

5.3.3.2 Video mode limitations

5.3.3.2.1 Video mode background

A video mode is described by 9 parameters: the dot clock speed, four for the horizontal timing and four for the vertical timing. These parameters characterize the number and shape of the displayed pixels, the size of the horizontal and vertical blanking, and the size of the horizontal and vertical sync pulses. The dot clock speed is the rate at which pixels are output.

The four values for the horizontal timing specify the format of a single horizontal line. The first value, *hdisp*, is the number of pixels on a horizontal line that contain display

data. The second value, *hsyncstart*, is the number of pixels into the horizontal line at which the horizontal sync pulse beings. The third value, *hsyncend*, is the number of pixels into the horizontal line at which the horizontal sync pulse ends. The fourth value, *htotal*, is the total number of pixels in a horizontal line. The difference between *hsyncend* and *hsyncstart* is the width of the horizontal sync pulse. The value of *htotal* less the width of the sync pulse and *hdisp* is the amount of horizontal blanking.

The four values for the vertical timing specify how horizontal lines stack together to fill a single frame. The first value, *vdisp*, is the number of horizontal lines that contain display data. The second value, *vsyncstart*, is the number of lines into the frame at which the vertical sync pulse beings. The third value, *vsyncend*, is the number of pixels into the vertical frame at which the vertical sync pulse ends. The fourth value, *vtotal*, is the total number of horizontal lines in a complete frame. The difference between *vsyncend* and *vsyncstart* is the width of the vertical sync pulse. The value of *vtotal* less the width of the sync pulse and *vdisp* is the amount of vertical blanking.

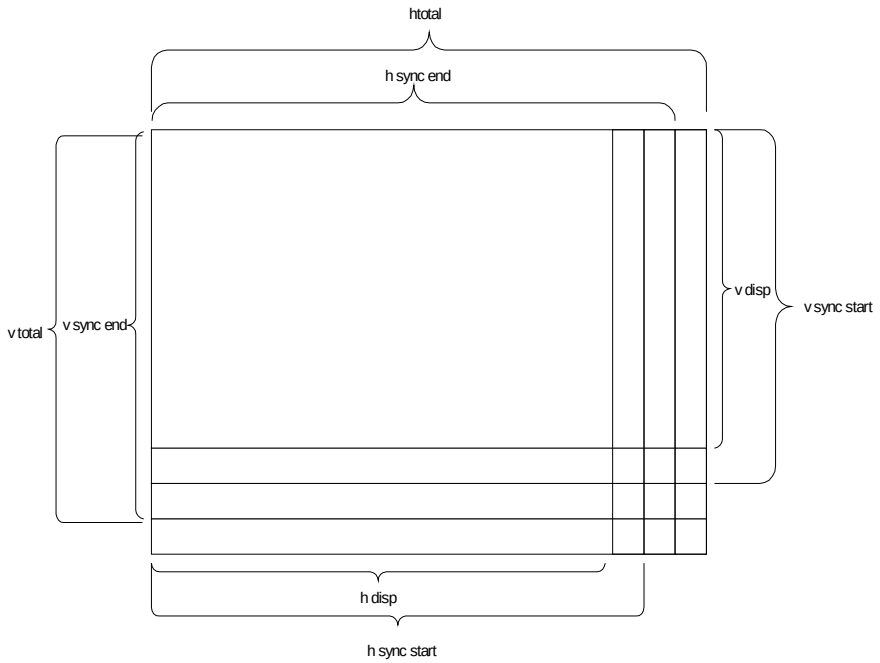



**Figure 11: Video mode parameters.**

5.3.3.2.2 Limitations

The drivers for the Quadro FX 3000G, as well as for most video cards, have restrictions on the values for video mode parameters. All horizontal timing values must be multiples of 8. Also, *htotal* must be greater than *hsyncend*, which must be greater than *hsyncstart*. This means that the minimum horizontal blanking and the minimum horizontal sync pulse width are both 8 pixels. The vertical timing parameters are not restricted to multiples of 8 but do have analogous requirements that *vtotal* be greater than *vsyncend*, which is greater than *vsyncstart*. Additionally, the maximum vertical sync pulse width is 16 lines. The maximum value for *hdisp* is 4,096 and the maximum value for *vdisp* is 2400.

### 5.3.4 **Constructing 18 synchronized data outputs**

The Quadro FX 3000G can provide a maximum of 6 parallel data outputs by using each red, green, and blue channel in a dual-head configuration as a separate data output. We therefore need a minimum of 3 video cards to provide the necessary 18 outputs. Since we will also use the video cards to compute holograms, there are advantages to using more video cards with each card outputting fewer channels. For example, we could use a single-head configuration for 6 video cards, only the red channel with a dual-head configuration for 9 video cards, or only the red channel with a single-head configuration for 18 cards. For monetary reasons and to prove that we can drive Holovideo with as few PCs as possible, we chose to use only 3 video cards. The 3 video cards are connected together in a linear chain using the video cards' frame lock feature. Since we only need to synchronize the cards among each other and not to an external source, the genlock feature of the cards is not used.

### 5.3.5 **Video mode**

Holovideo has a horizontal data line length of 256K pixels plus a 0.9 ms horizontal blanking period. For a 2.4 ms active period, this gives a total of 360,448 samples per horizontal line. For each of the 18 channels, there are 8 vertically stacked lines, followed by a vertical sync pulse period and a vertical blanking period whose combined times are equal to the length of two lines, for a total of 10 vertical lines.

The video card limits the maximum horizontal line display size to 4,096 pixels. We therefore cannot output one line of Holovideo input as a single line of PC output. Our solution is to use multiple video card lines to supply each Holovideo line. Using this

method, however, we cannot rely on the video card's horizontal blanking to provide for Holovideo's horizontal blanking. To get around this, we expand our video mode's display pixels to include pixels for Holovideo's horizontal blanking time and write zero values to those pixels.

Our target number of display samples per Holovideo line is 360,448. For various reasons, we would like the video card horizontal line length to be a multiple of 1K. To achieve the desired number of samples, we can either use 176 vertical lines with a 2,048 sample line length or 88 vertical lines with a 4,096 sample line length. As we'll show later, changing the number of vertical lines is more difficult than changing the horizontal line length. Since 4,096 is the maximum line length the drivers will accept, to allow for future increases in the number of samples in a hololine, we chose to use 176 vertical lines at 2,048 samples per line. This configuration gives 128 vertical lines worth of fringe pattern data and 48 lines worth of horizontal blanking values.

Each horizontal line on the video card also includes pixels for the horizontal sync period and the horizontal blanking period. Since the video card restricts both the horizontal sync period and the horizontal blanking period to be a minimum of 8 pixels each, we must subtract the 16 unused pixels from each horizontal line's display size. We arrive at values of *hdisp*=2032, *hsyncstart*=2032, *hsyncend*=2040, and *htotal*=2048. This means that 16 out of every 2,048 pixels fed to Holovideo will be blank, amounting to a loss of less than 1 percent of the data values. In practice, the missing pixel values do not produce any visible artifacts.

Each line of Holovideo input requires 176 horizontal lines of PC output. Each channel drives 8 lines of Holovideo input so we need a total of 1,408 lines of output data on the video card. We need a vertical sync period and vertical blanking period that sum to 2 lines of Holovideo input or 352 lines of video card output. We use the minimum value of 8 lines for the vertical sync period. Ideally, we would use 344 lines of vertical blanking to fill the remaining blanking time required by Holovideo. However, when frame lock is enabled on the Quadro FX 3000G, the video drivers reconfigure the video mode and remove the vertical blanking period. To get around this, we use the minimum value of 8 pixels for the vertical blanking period and instead add an additional 344 lines of data pixels. To match what Holovideo is expecting, we add the additional lines at the beginning of the display pixels and zero fill them. The video mode values are then *vdisp*=1744, *vsyncstart*=1752, *vsyncend*=1760, and *vtotal*=1768. When the drivers remove the vertical blanking period of 8 lines, the total number of lines is 1,760 as expected by Holovideo.

The value for the dot clock rate is chosen to make the vertical refresh rate 30Hz. When the drivers alter the video mode and remove the vertical blanking period, they also rescale the dot clock value to maintain the same vertical refresh rate. We therefore choose a value to achieve 30Hz with *htotal*=2048 and *vtotal*=1768. For an XFree86 modeline style input, the dot clock value is 108.626.

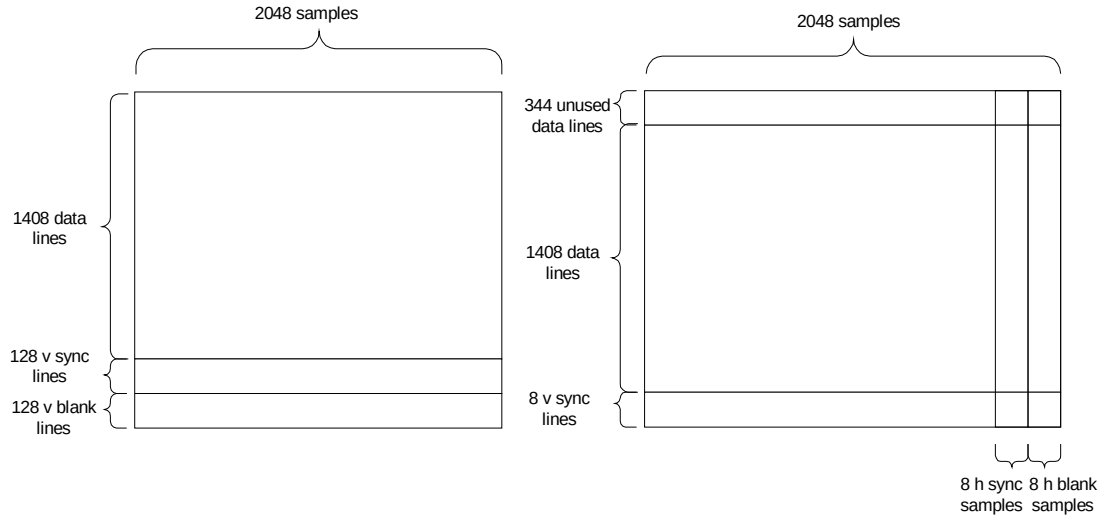



**Figure 12: Video mode diagrams for a single output channel. The first image is the ideal PC video mode. The second image is the video mode we actually use after the drivers remove the vertical blanking.**

The complete XFree86 modeline is:

ModeLine “2048x1760” 108.626 2032 2032 2040 2048 1752 1752 1760 1768 +hsync +vsync

5.3.5.1 Hologram framebuffer data format

As per the discussion above, the framebuffer for each channel is 2,032 samples wide and 1,752 lines tall. To write a fringe pattern to the framebuffer for display, the first 344 lines must be black. The next 176 lines contain the first three hololines output by the framebuffer (the first line in the red channel, the second in green, and the third in blue). The following 176 lines contain the second three hololines, and so on for a total of 8 hololine triplets.

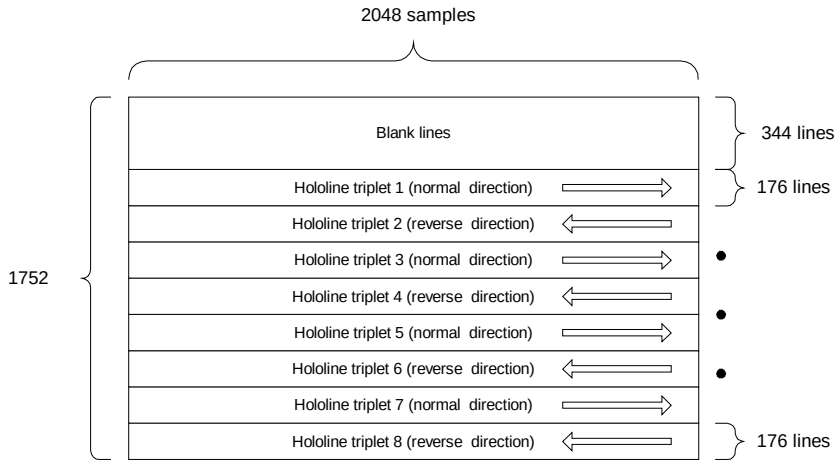



**Figure 13: The format for writing a holographic fringe pattern to the framebuffer.**

Each channel of each 176-line tall hololine triplet outputs a single long holographic fringe pattern. The first 128 lines contain fringe pattern data and the last 48 lines are zero filled. To accommodate the display's boustrophedonic scanning pattern, alternating triplets are stored in normal and reverse order. For a normal direction triplet, the first sample of the fringe pattern is stored at the first location and the last sample at the last location. Fringe pattern samples are laid down left to right and broken up over the 176 vertical lines from top to bottom. For a reverse direction triplet, the first sample of the fringe pattern is stored at the last location and the last sample at the first location. Fringe pattern samples are laid down right to left and broken up over the 176 vertical lines from bottom to top.

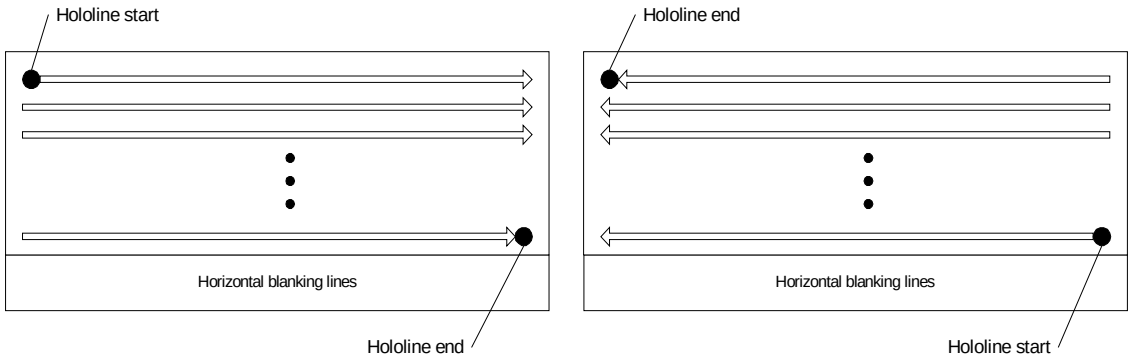



**Figure 14: The format of each hololine in the framebuffer. The first image is the format for a normal direction hololine. The second image is the format for a reverse direction hololine.**

Because of our inability to remove the horizontal blanking, the framebuffer is 16 samples narrower than it should be, resulting in a loss of 2,048 samples per hololine. To write a fringe pattern to the framebuffer, we assume the framebuffer is the correct 2,048 samples wide and drop the 16 samples per horizontal line that are not available for writing. In contrast to dropping the last 2K samples, our method has several benefits. First, it preserves our ability to fit exactly two 1,024-sample hogels on each horizontal line. Second, if the missing samples were to produce visible artifacts, the image would appear to have a black grating in front of it as opposed to appearing as if the image were cut into vertical slices and horizontally separated (we do not introduce horizontal stretching and discontinuities). Third, dropping the last 2K samples would have a noticeable impact on the width of the view zone.

## 5.4 Horizontal sync input

### 5.4.1 Requirements from Holovideo

To supply the horizontal sync signal, the system must output a TTL rising edge at the beginning of each hololine.

### 5.4.2 **Driving the horizontal sync input**

The horizontal sync output of the video cards is not suitable to drive Holovideo directly. The video card horizontal sync pulses once per video card line, or 176 times per Holovideo line. To convert the video card horizontal sync signal to a signal suitable to drive Holovideo, we built a simple TTL circuit that outputs one rising edge for every 176 rising edges on the input signal. The horizontal sync signal from a video card is used as the input to the converter circuit and its output is sent directly to Holovideo's horizontal sync input. Refer to Appendix A for a schematic of the horizontal sync converter circuit.

## 5.5 **Vertical sync input**

### 5.5.1 **Requirements from Holovideo**

To supply the vertical sync signal, the system must output a TTL rising edge at the beginning of the Holovideo line and the output must return to low before the end of that line.

### 5.5.2 **Driving the vertical sync input**

The vertical sync signal from any of the video cards is suitable to directly drive Holovideo's vertical sync input.

# 6 USING PCS TO COMPUTE HOLOGRAMS

## 6.1 Introduction

In this chapter, we discuss using our new PC based system to compute holograms. We begin by examining the requirements imposed by Holovideo and by the algorithm to compute diffraction specific holographic stereograms. We then discuss previous work relevant to computing holograms on a PC based system. Next we discuss our system's computational capabilities, including a discussion of the overall computing architecture, the capabilities of the GPU, and how our system compares to the Cheops based system. Finally, we describe a prototype implementation to compute holograms in real-time.

## 6.2 Requirements

### 6.2.1 Requirements from Holovideo

Strictly speaking, our use of the video cards' framebuffers decouples the requirements of hologram computation from the requirements of the Holovideo display. However, our goal is to compute content at the display's maximum rate of 30 frames per second. Each frame contains 18 channels of 8 lines at 256K data samples per line, for a total of 36MB per frame. To deliver each frame, our system must compute and write to the framebuffer a 36MB fringe pattern. At 30Hz and with 18 parallel channels, this requires that our system be able to write 60MB/s/channel worth of data to the framebuffer.

### 6.2.2 **Requirements from hologram computation algorithms**

The algorithm for computing a diffraction-specific hologram for our new viewing geometry is as follows: First, compute 32 basis fringes that diffract light evenly into 32 angular segments subtending a total of 30 degrees. Each Hololine contains 256K samples and each hogel is 1K samples long, so each hololine now contains 256 hogels. Second, compute 32 images from the 32 viewing angles, each 256x144 pixels. Divide each hololine into 256 non-overlapping adjacent segments 1,024 pixels in length. The *i*th hogel is the normalized sum over the 32 views of a viewing angle's basis fringe multiplied by the *i*th pixel of that viewing angle's rendered image.

In our system, all data inputs to Holovideo are 8-bit values. To preserve this level of accuracy, each sample of the basis fringes and of the rendered views is computed to 8-bits of precision. In order to compute the normalized sum over 32 views with 8-bits of accuracy, we need at least 13 bits of accuracy to accumulate the sum over view pixels multiplied by basis pixels.

## 6.3 **Previous work**

Using the GPU for purposes other than its intended role in conventional computer graphics is not a new concept. It has been an area of active research for a number of years. There is an online group called General-Purpose computation of a GPUs (available at *http://www.gpgpu.org*) dedicated to using the GPU for general purpose computing. Venkatasubramanian gave a talk on the trends of using the graphics card as a high-performance stream co-processor [20].

### 6.3.1 **Accumulation buffer based holographic stereogram computation**

Lucente's 1996 paper titled "Rendering Interactive Holographic Images" introduced an algorithm to compute holograms that uses the video card's hardware accelerated accumulation buffer and texturing units [18]. The algorithm is specifically aimed at computing diffraction specific holograms but applies to most methods of calculating holographic stereograms. The texturing units of the GPU are used to modulate basis fringes by view image information. The modulated basis fringes are written to the framebuffer and summed together using the accumulation buffer. We model our implementation after Lucente's accumulation buffer based algorithm.

### 6.3.2 **High precision computing with commodity video cards**

Using an accumulation buffer based algorithm to sum basis fringes was not an option on commodity hardware until recently. PC video cards lacked hardware accelerated accumulation buffers. Using a software emulated accumulation buffer to perform the basis fringe summation would result in slower performance than running the entire computation on the CPU. As it turns out, texture units can also be used to sum values. However, until recently, they were restricted to 8 bits of precision. Petz and Magnor presented an algorithm that splits the necessary basis fringe summations into a hierarchy of texture summations [8]. The hierarchy is organized in such a way the resultant summation has the level of accuracy necessary to calculate holograms. However, newer video cards do in fact have hardware based accumulation buffers. Additionally, newer cards allow textures to be stored and manipulated with high precision. Both of these developments make this algorithm irrelevant to our project.

## 6.4 **Platform computational capabilities**

### 6.4.1 **System overview**

The system to drive Holovideo consists of three PCs, each with a Quadro FX 3000G. Since the video cards can do most of the hologram computation, the PCs are modestly equipped with the most inexpensive available components. Each PC has an Intel Pentium 4 processor clocked at 2.0GHz, 256MB of DDR SDRAM, and an ATA hard disk. The Quadro FX 3000G is connected to the processor over an AGP 8x interface. Each Quadro FX 3000G has 256MB of local memory. Schematically, the pertinent architecture is shown in Figure 15.

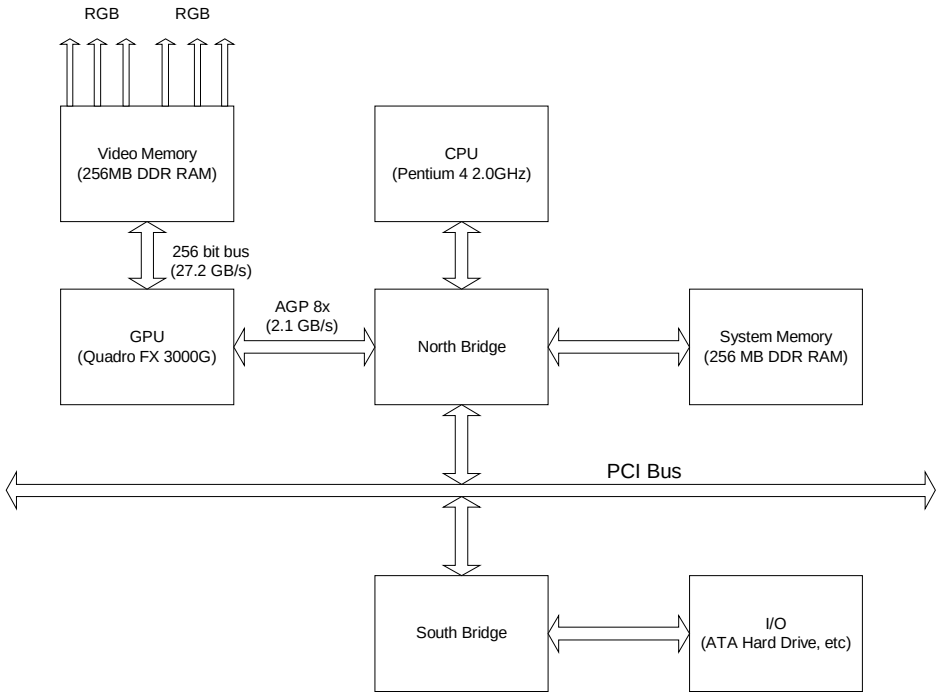



**Figure 15: Computing hardware architecture of each PC.**

The CPU is connected to the main system memory over a 533MHz bus (can be increased to 800MHz with a modest hardware upgrade). The GPU is connected to its local memory

over a 27.2GB/s bus. The GPU and CPU are connected over an AGP 8x bus that transfers data at 2.1GB/s. The system can access permanent storage (hard disk) at a maximum rate of roughly 100MB/s.

#### 6.4.1.1 Bandwidth considerations

To compute a hologram on our system, we load a 3D model from permanent storage into main memory. We then use the CPU to load static 3D geometry and GPU program instructions into video memory. From there, the GPU calculates a holographic fringe pattern and writes it to video memory. To achieve interactivity and animation, the CPU sends geometry updates and transformations to the GPU and video memory.

Since Holovideo updates its image at 30 frames per second, our target frame rate is at least 30 frames per second. We are limited by the maximum bandwidth of the busses that connect the pertinent components in our system. From the requirements section, we know that our target frame rate yields a target bandwidth to the framebuffer of 60MB per second per channel. Since each video card is responsible for 6 channels, we need a minimum of a 360MB/s connection from the GPU to the framebuffer. This lies well within the maximum bus rate of 27.2GB/s. To achieve animation and interactivity with our scene geometry, we must transmit geometry updates over the AGP bus. Geometry updates can be transmitted either by performing geometry transformations on the CPU and sending the updated geometry information to the GPU or by sending procedurally defined transformations to the GPU with which it can update the scene geometry stored in the local video memory. The maximum amount of updated data or procedurally

defined transformations that can be sent per frame at 30Hz from the CPU to the GPU is about 71MB.

### 6.4.2 **nVidia Quadro FX 3000G capabilities**

#### 6.4.2.1 Programmable vertex and fragment processors

##### 6.4.2.1.1 Traditional OpenGL pipeline background

The OpenGL rendering pipeline takes a stream of vertices from memory through a vertex processor that performs per vertex operations. The output from the vertex processor is then rasterized to screen elements called fragments. The fragment stream is run through a fragment processor that performs per fragment operations. Finally, the output of the fragment processor is written to the framebuffer. [13] [15]

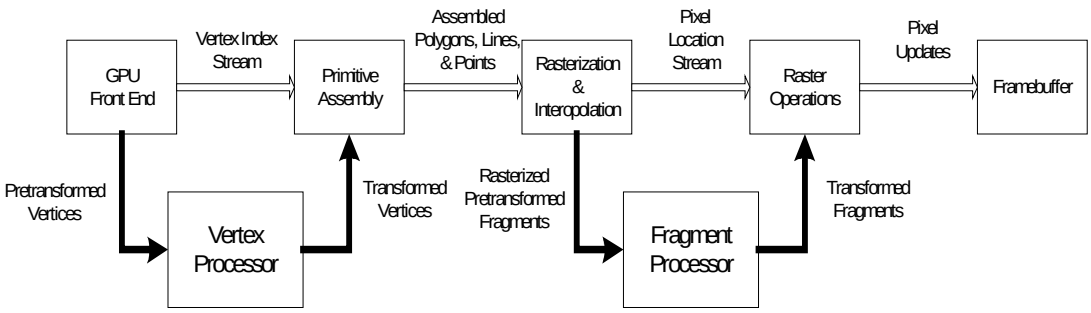



**Figure 16: Model of a GPU and its pipeline. [16]**

The vertex processor performs tasks such as model-space to screen-space projections and per vertex lighting. The rasterizer is responsible for interpolating per vertex values such as color and texture coordinates to their values at screen pixel locations. The fragment processor performs per pixel tasks such as texture lookups and color blending.

##### 6.4.2.1.2 CineFX 2.0 Engine

In the traditional OpenGL pipeline, the vertex and fragment processors perform a series of fixed functions conditional only on a fixed set of state variables. In a modern GPU, including the Quadro FX 3000G, the vertex and fragment processors are programmable. They can accept custom code to perform custom operations. [14]

The CineFX 2.0 Engine is nVidia's proprietary graphics pipeline including its programmable vertex and fragment processors. The vertex processor can execute up to 65,536 instructions per vertex. The fragment processor can execute up to 2,048 instructions per fragment. Both are capable of performing computation using 12 bit fixed point precision, 16 bit IEEE floating point precision, or 32 bit IEEE floating point precision per channel.

6.4.2.2 Capabilities

The Quadro FX 3000G is equipped with a 16 bit per channel accumulation buffer. It is capable of applying 16 textures per pixel using 8 interpolated texture coordinates per pass. All calculations can be performed using full 128-bit floating point precision (32 bits per channel). Each channel of textures, pbuffers, and framebuffers can be calculated to and stored with the standard 8 bit precision, 12 bit fixed point precision, 16 bit floating point precision, or 32 bit floating point precision.

In terms of raw performance, the Quadro FX 3000G can move 100 million triangles/second through its vertex processor and 3.2 billion texels/second through its fragment processor. The GPU is connected to a 256MB DDR memory bank through a 256-bit wide memory bus that is capable of 27.2 GB/second data transfers. [12]

## 6.5 Using the GPU to compute holograms

The Quadro FX 3000G has the ability to perform all of its computations using 32-bit floating-point precision and to retain that precision when storing values in texture or framebuffer memory. Our new system therefore meets the 13-bit precision requirement to compute a diffraction specific holographic stereogram. The OpenGL texture modulation feature is sufficient to module a basis fringe with a weight value from a view image. Either the accumulation buffer or a custom fragment program is sufficient to sum and normalize the modulated basis fringes into a complete holographic fringe pattern.

### 6.5.1 Comparison to Cheops

The model of computation on our new system is similar to the one it is replacing. We have a general purpose CPU connected to high performance stream processors through a relatively slow bus. Both the CPU and stream processors have a local memory bank. To efficiently compute holograms, we must limit the amount of data that must be sent between the CPU and the stream processors. However, the computational capabilities of the video card's stream processors make our system much more efficient than the Cheops based architecture.

To compute a holographic stereogram with the Cheops system, we compute the view images on the general purpose CPU (SGI) and reorganize them into Hogel-Vectors. We then send the Hogel-Vectors to Cheops where they are combined with the basis vectors to construct a holographic fringe pattern that is written to the framebuffer. In this system, all of the information contained in the view images must be transmitted over the slow bus

to the stream processors. In our new system, the video card's stream processors are capable of generating the view images and saving them to local memory. The video cards can then modulate the view information with the basis fringes and write a holographic fringe pattern to the framebuffer. The only information that needs to be sent over the slow bus is data independent program instructions for the video card.

### 6.5.2 **Accumulation buffer algorithm to compute holograms**

We give a simple accumulation buffer based algorithm to compute holographic stereograms [18]. Each of the three PCs in the Holovideo system runs a process responsible for drawing holographic fringe patterns to the framebuffer. A fourth PC runs a server process to synchronize and control the three PCs in our system. The additional PC runs a user interface on a standard LCD and accepts mouse and keyboard input for interactive content.

#### 6.5.2.1 Rendering synchronization

The three processes are synchronized with a simple client/server model. A fourth PC runs a server process that listens for connections from the three rendering processes (the clients). In addition to providing extra user interface capabilities, running the server on a fourth PC instead of on one of the content machines has the advantage of ensuring that the communication time between client and server is symmetric for all clients.

The sequence to render a frame is as follows. Each client sends a message to the server when it is ready to render. When all three clients are ready to render, the server sends a message to each client to render a frame. Each client responds with an acknowledgement

message when the frame has been rendered. When all three clients have responded, the server sends a message to each client to swap buffers and display the newly rendered frame. Each clients responds by telling the server it is again ready to render after the buffers have been swapped.

To enable animation and user interactivity, the sequence to update the scene geometry is as follows. When all three clients are ready to render, the server sends a message to each client to transform its scene graph along with the relevant information about what transformation to make. The clients apply the transformation, render a new frame, and respond with an acknowledgement message when the frame has been rendered. When all three clients have responded, the server sends a message to each client to swap buffers. The clients respond to tell the server they are again ready to render.

#### 6.5.2.2 Hologram computation

An outline of a simple algorithm to be run by each of the rendering processes is given below. Note that the process is responsible for writing a fringe pattern to both output framebuffers on its PC.

```
Compute 32 basis fringes on the CPU
Create a 1D textures for each basis fringe
For each frame
    Render 32 views of the scene geometry into the framebuffer and create a 2D
    texture out of each view
    Clear the accumulation buffer
    For hololine = 1 to 24
        For view = 1 to 32
            Clear the framebuffer
            Bind basis fringe[view] to texture 1
            Bind view image[view] to texture 2
            For hogel = 1 to 256
                Write texture 1 modulated by texture 2 to the correct
                location and to the correct color channel in the
                framebuffer
            Add the framebuffer contents to the accumulation buffer
    Divide all values in the accumulation buffer by 32
```

Write the contents of the accumulation buffer to the framebuffer

6.5.2.2.1 Optimizations

We make a number of optimizations to this algorithm. First, we pack all of the 1D basis fringe textures into a single 2D texture where each row of pixels in the 2D texture is a single basis fringe. This eliminates the need to unbind and bind a new basis fringe texture between views. Second, we pack all of the view images into a single 2D texture. We also render to a pbuffer that can be used directly as a texture instead of reading data from the framebuffer into a texture. This eliminates the need to unbind and bind the view image textures between views and eliminates the need to copy pixels from the framebuffer to texture memory. Third, we store the vertex and texture coordinate data needed to construct the fringe pattern in a display list to minimize the amount of data transferred over slow busses. Fourth, we pre-compute the packing of three hololines into the three color channels of a single output pixel. We do this at the stage of preparing the view textures. We render the 32 views to the red channel of the framebuffer (a pbuffer). When then shift the viewport up one horizontal line and render the same 32 views to the green channel. Finally, we shift the viewport up one more line and render the views to the blue channel. To compute the fringe pattern, we render 8 lines using all three of the color channels. (For modest scene geometry, rendering three times as many views is faster than reading data from the framebuffer to shift it and write it back to the framebuffer.) This way, we can modulate view images and write out fringe patterns for three hololines in one parallel step. This optimization reduces the number of texture operations that need to be performed as well as reduces the amount of data that needs to be written to the framebuffer with the color mask enabled.

# 7 RESULTS

## 7.1 Image quality

### 7.1.1 Holovideo display artifacts

The Holovideo display exhibits several artifacts neither attributable to nor correctable by our framebuffer system. The design of the display yields a number of visible scanning artifacts. Since the scanning system lays down blocks of 18 horizontal lines at a time, most scanning artifacts are situated about the blocks of horizontal lines. A slight difference in speeds between the forward and reverse scans produce image discontinuities at the boundary between blocks of lines laid down in opposite directions, called boustrophedonic scan errors. The settling time between steps of the vertical scanning mirror produces slight gaps between blocks near the right and left edges of the image, called vertical scanning errors. The use of two orthogonal scanning elements (a horizontal and a vertical mirror) produces a pinching of the image in the vertical direction towards the right and left edges of the image, called bow scan distortion. For more details and example images, see St.-Hilaire's Ph.D. thesis [1].

The display also has a number of correctable image quality issues. A small chunk is missing from one of the horizontal scanning mirrors resulting in a small portion of each block of 18 horizontal lines not being imaged. The Bragg cells are not perfectly aligned, leading to a difference in brightness between the lines laid down on forward and backward scans. Misalignment also results in large gaps between the blocks laid down

on forward and backward scans. Finally, the RF processing hardware is not tuned correctly which results in a few missing horizontal lines.

### 7.1.2 **Comparison with Cheops images**

The images in Figure 17 show photographs of a hologram of a solid uniformly colored plane that mostly fills the view zone of the display. The left image is from Holovideo driven by Cheops and the right image is Holovideo driven by our new PC system. This first image elucidates most of the display artifacts not attributable the framebuffer. The image should be rectangular but is pinched towards the edges do to bow scan distortion. The striped appearance due to a difference in brightness between the forward and backward scans is a problem with alignment. The gaps between the lines drawn by forward and backward scans are another optical alignment problem. We are only interested in the image quality properties that are due to the framebuffer. We therefore ignore the artifacts discussed above and focus on comparing the images from the two different framebuffer systems.

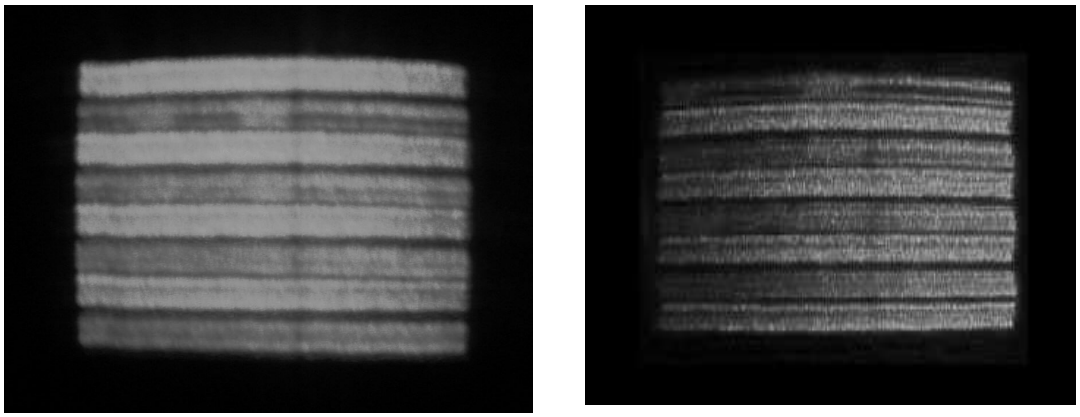


**Figure 17: Photograph of the output of a hologram of a solid rectangular plane. The first image is Holovideo driven by the Cheops framebuffer. The second image is Holovideo driven by the new PC based system.**

#### 7.1.2.1 Data input synchronization errors

By examining the left and right edges of the plane in Figure 17, we see the obvious artifact of our framebuffer. The edge should be straight but appears jagged in the image produced by the PC system. To elucidate the problem, we made a hologram of three vertical lines, one near the left edge of the display, one near the right, and one in the middle, shown in Figure 18. The boustrophedonic scanning errors of the display introduce discontinuities in vertical lines at the boundaries between forward and backward scans. However, the discontinuities in our images are not isolated to these locations. That fact combined with Cheops behavior on these same test holograms shows that this artifact is attributable to our framebuffer system.

After carefully examining test images and the outputs of the video cards with an oscilloscope, we determined that this image artifact is due to the data inputs not being perfectly synchronized. The frame lock feature of the nVidia Quadro FX 3000G under Linux is not accurate to the degree we need to get perfect image quality. Since the frames are not exactly synchronized, horizontal lines start at slightly different offsets, making vertical lines appear jagged. This is consistent with the observation that groups of lines from the same video card do not suffer from vertical discontinuities.

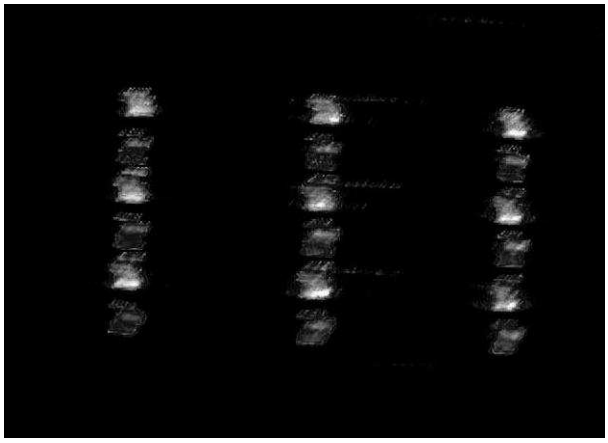


**Figure 18: Photograph of the output of a hologram of three vertical lines driven by the PC based system.**

7.1.2.2 General comparison

The other obvious image artifact to look for is periodic darkened vertical bands due to our inability to remove the horizontal blanking from the PC video mode, as reported by Lucente for a similar video configuration with 4 percent of the samples missing in an older display [18]. Several viewers examining numerous test images were unable to perceive any errors of this type. Our modification of the horizontal blanking period had no perceivable effects on the linearity of the horizontal scan, also determined empirically. The only remaining difference between the two framebuffers systems is the hardware itself. The nVidia video cards use high quality DACs and do not introduce any noticeable line noise or other artifacts.

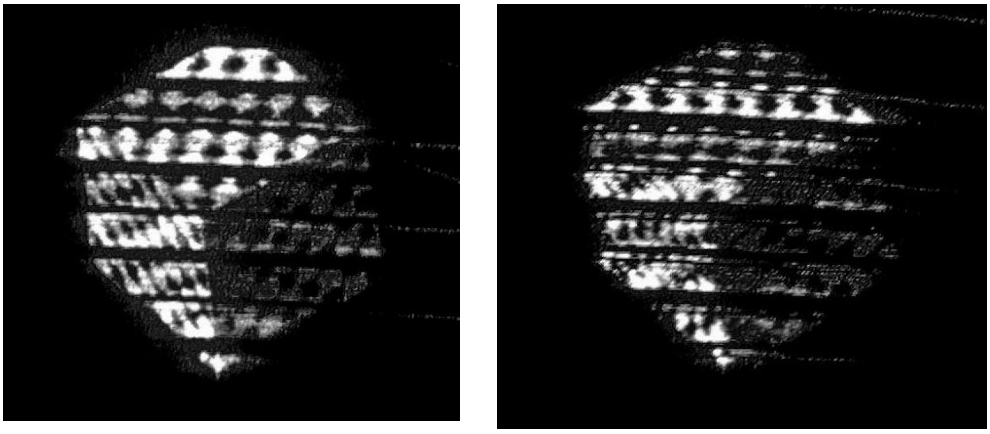


**Figure 19: Photograph of the output of a hologram of a textured cube. The first image is Holovideo driven by the Cheops framebuffer. The second image is Holovideo driven by the new PC based system.**

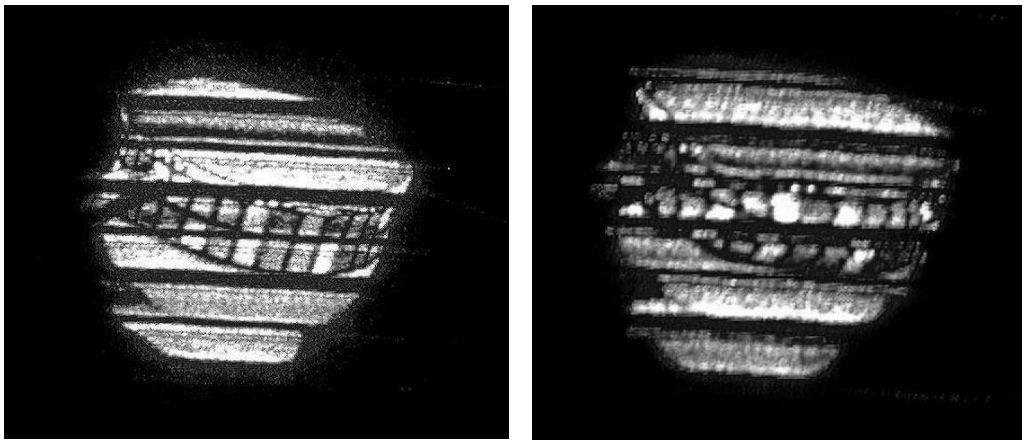


**Figure 20: Photograph of the output of a hologram of a teacup. The first image is Holovideo driven by the Cheops framebuffer. The second image is Holovideo driven by the new PC based system.**

To compare the overall image quality and to examine the impact of the synchronization errors on more realistic holograms, we give images of a textured cube and a textured teacup in Figure 19 and Figure 20 respectively. The vertical discontinuities introduced by the synchronization errors are difficult to perceive on models without clean vertical lines. Errors are even more difficult to perceive when viewing animated holograms.

Overall, the images produced by Cheops and the PC system are very similar in quality and difficult to distinguish.

## 7.2 **Hologram computation**

The focus of this thesis was building a platform to serve as a framebuffer for Holovideo that is also capable of computing holographic content in as close to real-time as possible. Though not strictly within the scope of this project, we implemented the scheme described in 6.5.2 to compute diffraction-specific holographic stereograms. Our implementation delivers dynamically animated interactive content using modest scene geometry at about 2 frames per second.

### 7.2.1 **Computation speeds**

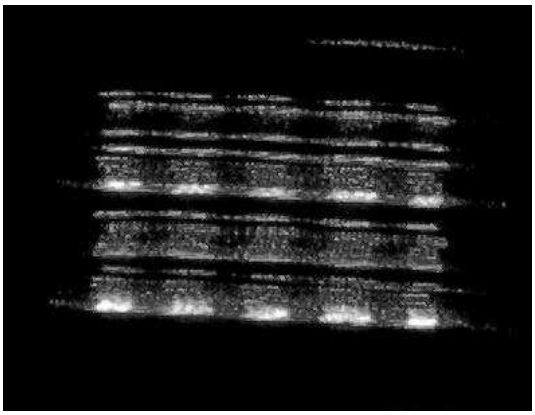


**Figure 21: Photograph of an animation of a rotating textured cube.**

The image in Figure 21 is a photograph of an animation of a rotating textured cube. The model consists of 12 single-textured triangles. The scene uses a total of four lights and refreshes at about 2 frames per second.

The computation of any stereogram can be divided into two parts: computing the view data for the stereogram's view locations and constructing the stereogram from the view data. In our case, computing the view data involves rendering 32 different views of the scene geometry. Call this the scene capture step. Constructing the stereogram involves modulating the 32 basis fringes by the view data and accumulating the modulated fringes. Call this the fringe computation step.

The complexity of the algorithm for the fringe computation step is not dependent on the input view images. It therefore always takes the same amount of time to perform. In our implementation, we measured the time to perform the fringe computation step to be about 0.45 seconds out of a total of about 0.52 seconds to compute the complete hologram. The scene capture step, however, is dependent on the complexity of the three-dimensional model being used. The total time to compute a fringe pattern is then the fixed time for the fringe computation step plus the variable time for the scene capture step.

OpenGL commands are executed on the GPU in an asynchronous pipelined fashion. It is therefore very difficult to accurately measure the breakdown of computation time for steps of an algorithm on the GPU. Our times were calculated by inserting a rendering barrier and flushing the pipeline. They are therefore not an entirely accurate picture of how the algorithm performs when it is not being timed.

# 8 FUTURE WORK

## 8.1 Replace RF hardware

The most unreliable part of the Holovideo display is the mixer in the RF signal processing hardware. The RF hardware serves two roles. First, it mixes the data inputs with a sine wave to shift the data to the frequency range required by the Bragg cells. Second, it filters and amplifies the input signals before passing them to the Bragg cells. Though we need the second filtering and amplification stage, we can adjust the output from the video cards to match the frequency required by the Bragg cells in order to remove the mixing step from the RF hardware.

The Bragg cell frequency range is 50-100MHz. To accommodate this, the RF hardware uses a mixer to modulate the lower sideband of the data inputs with a 100MHz sine wave. The output from the video cards is about 108MSamples/second. From the Nyquist theorem, this means the output of the video cards can accommodate a signal of at most 54MHz. In order to accommodate the 100MHz carrier frequency of the Bragg cell, we need to output 200Msamples/second.

To achieve 200Msamples/second, we can expand our horizontal lines to be the maximum value of 4,096 samples per line. This gives us a total of 216Msamples/second, well above the value we need to eliminate the mixer. Doubling the number of samples in each horizontal line effectively halves the size of the horizontal sync pulse. In order for the

horizontal sync converter circuit to function correctly, we also need to double the size of the horizontal sync pulse to 16 samples (the capacitance of the breadboard begins to introduce errors at higher speeds).

To compute holograms we need to add an additional post-processing step of modulating the fringe pattern with a 100MHz sine wave. To do this, the computation system writes a normal 2,048 by 1,408 sample fringe pattern to the framebuffer and makes a texture using the data in the framebuffer. We then draw a single rectangle that maps to the full 4,096 by 1,408 sample framebuffer mapped with two textures. The first texture is the fringe pattern that was read from the framebuffer applied in decal mode. The second texture is a pre-computed texture containing a 100MHz sine wave applied in modulation mode.

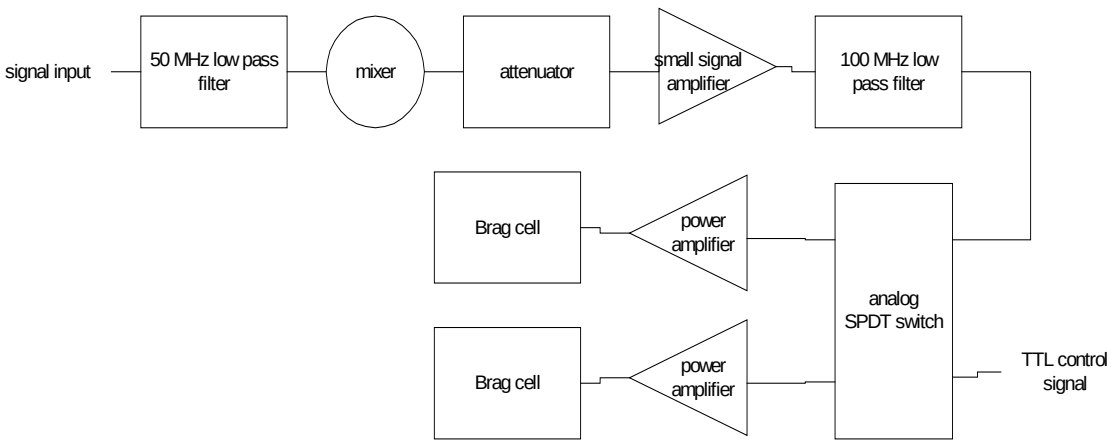



**Figure 22: Block diagram of the RF circuit for each data channel.**

A block diagram of the RF circuit for each data channel is show in Figure 22. Since we are providing data input in the correct 50-100 MHz range, we no longer need the mixer or the 50 MHz low pass filter.

## 8.2 **Hologram Computation**

Computing holograms is an essential step to make use of our new platform. Computing higher quality holograms at higher frame rates is critical to the success of holographic video displays.

### 8.2.1 **Optimizations**

At the time this thesis was written, our code to compute diffraction specific holograms using the accumulation buffer runs at about 2 frames per second. This figure can most likely be improved with a better implementation. We need to carefully analyze the configuration of our OpenGL pipeline to remove bottlenecks. We need to take full advantage of the OpenGL API to ensure that each step is efficient. We also need to examine the possibility of using custom vertex and fragment programs to improve performance [13]. Additionally, we need to examine the possibility of more efficient ways to compute the view images. For example, it may be possible to develop a hardware-accelerated implementation of Halle's multiple viewpoint rendering (MVR) algorithm [10].

### 8.2.2 **RIP/RIS holograms**

Reconfigurable Image Plane (RIP) holograms are a recently developed alternative to Lucente's diffraction-specific holographic stereogram [4][6]. Conventional stereograms

project *n* views in *n* collimated directions by assembling a hologram from an *n*x*m* matrix of hogels. Conventional holograms are limited in several regards. Particularly, the hologram plane acts as the image plane and view sample boundaries are collocated with basis fringe boundaries. RIP/RIS holograms do not rely on an *m*x*n* matrix of hogels, but instead use one or a set of parallax-modulated elemental fringes with translated, partially overlapping footprints on the hologram surface to reconstruct one or more image planes or surfaces. Each surface is comprised of a set of image primitives that are reconstructed at a location offset from the hologram surface.

Computing RIP holograms is similar to computing Lucente's holograms. We just replace the basis fringes with a single fringe that focuses light at a single point in space, overlap the hogels on the hologram plane, and construct a parallax view vector instead of using a single view weight. Instead of using basis fringes to direct a certain intensity of light in a given direction, RIP holograms use a single fringe that directs every point in the horizontal extent of the hogel to a single point in space. Since this fringe is repeated for each hogel, the hologram is built up as an image of single points focused at a plane in front of the display. The RIP algorithm is extensible to another type of hologram, the Reconfigurable Image Sheet (RIS) algorithm. Instead of using a single fringe pattern that focuses light at a fixed distance from the display, the algorithm selects from a number of fringe patterns to focus light at a variable distance from the display. This allows the image to be focused at an arbitrarily shaped sheet in space such as a cylinder or the surface of the scene model, producing higher quality images.

RIP/RIS holograms have a number of qualities that lead to better image quality than older computation algorithms. The elemental fringes are pre-windowed and laid down on the hologram plane in a partially overlapping fashion to minimize phase-discontinuities at their boundaries. The parallax view sample vector by which an elemental fringe is modulated can be smoothly interpolated to avoid discontinuities in the parallax information. Since the image surface is focused off the hologram plane, the image plane of a RIP hologram has a higher spatial resolution and has a higher scene parallax than that of a conventional stereogram.

RIP holograms are quick to compute and adapt well to hardware acceleration using modern video cards. The quality of the images they produce is also markedly better than Lucente's diffraction-specific holograms. The next step to improving the quality of the holograms we are computing is to develop a code to compute RIP and RIS holograms on our new system.

## 8.3 **Improving image quality**

### 8.3.1 **Remove horizontal blanking**

The horizontal blanking introduces 16 blank samples into every 2,048 samples of each output hololine. Although the blank pixels do not appear to introduce any visible artifacts, it is desirable to remove them. To do so, we need to work with nVidia to patch their drivers to allow zero horizontal blanking and if their hardware permits, a horizontal

pulse width that does not take away from the number of display pixels output per horizontal line.

### 8.3.2 **Improve genlock/frame lock**

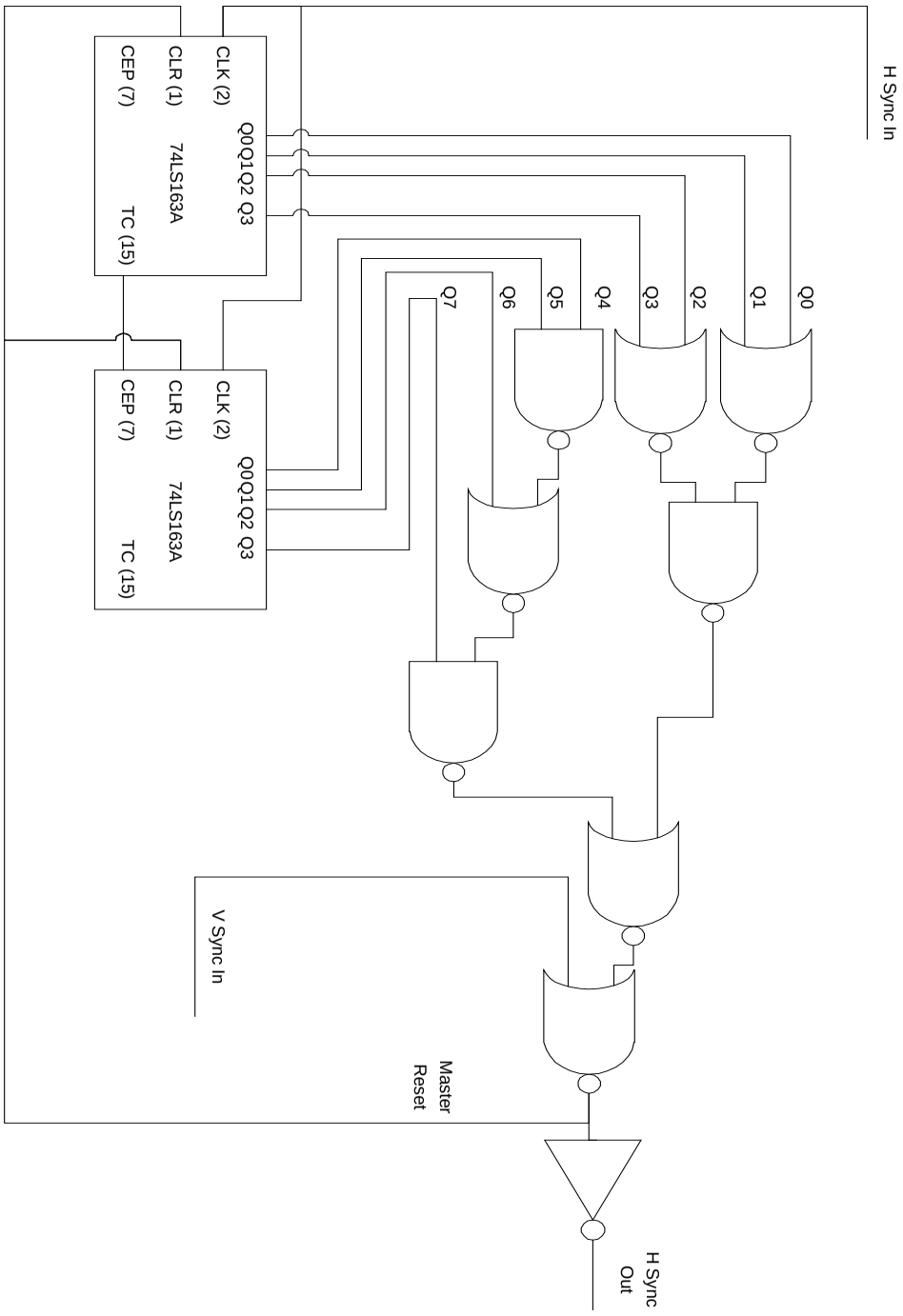
As mentioned in the results section, the genlock and frame lock features of the nVidia Quadro FX 3000G under Linux are not accurate enough to give perfect image quality. Since the frames are not exactly synchronized, horizontal lines start at slightly different offsets. Vertical lines therefore appear jagged. To improve the image quality, we need to improve the accuracy of the synchronization of our output channels. To do this, we either need to work with nVidia to improve our system setup, to improve the quality of the Linux drivers, or to improve the quality of frame lock in future video cards.

# 9 CONCLUSION

Recent advances in PC technology, particularly in the video card industry, have made it possible to drive a holographic video display in real-time with inexpensive off the shelf hardware. Our PC based system provides a framebuffer that delivers output with image quality comparable to the Cheops framebuffer it is replacing. It provides the additional benefit of more computational power, resulting in higher frame rate video using better quality holograms of more complicated data sets. Reliance on off the shelf PC hardware ensures that our system is simple and inexpensive to maintain and that the exponential improvements in PC technology can be applied to producing real-time holographic content.

Our prototype implementation to compute diffraction specific holographic stereograms on our PC based system shows that it is capable of producing holographic video at interactive frame rates. Although our prototype implementation does not achieve the desired update rate of 30 frames per second, further optimizations and research in algorithmic advances, coupled with the raw computational power of a next generation video card when it becomes available will almost certainly yield full motion video frame rates. Our PC based content creation and delivery system pushes holographic video one step closer to moving out of the research arena and into the commercial market.

# 10 APPENDIX A: HORIZONTAL SYNC CONVERTER CIRCUIT



# 11 REFERENCES


[1] P. St.-Hilaire, *Scalable Optical Architectures for Electronic Holography*. Ph.D Thesis, MIT Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1994.

[2] M. Lucente, *Diffraction-Specific Fringe Computation for Electro-Holography*. Ph.D Thesis, MIT Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1994.

[3] W. Plesniak, "Incremental update of computer generated holograms". (Accepted for publication, Optical Engineering, June 2003).

[4] W. Plesniak and M. Halle, "Reconfigurable Image Plane (RIP) holograms". Manuscript currently under production.

[5] J. A. Watlington, M. Lucente, C. J. Sparrell, V. M. Bove, Jr., and I. Tamitani, "A Hardware Architecture for Rapid Generation of Electro-Holographic Fringe Patterns". *Proc. SPIE Practical Holography IX, 2406*, pp. 172-183, 1995.

[6] W. Plesniak, M. Halle, et al, "Holographic Video Display of Time-Series Volumetric Medical Data". (Accepted for publication Vis2003).

[7] P. Hariharan, *Optical Holography*, Cambridge University Press (1996).

[8] C. Petz and M. Magnor, "Fast Hologram Synthesis for 3D Geometry Models using Graphics Hardware". *Conference 5005 Practical Holography XVII*, 2003.

[9] D. Leseberg, "Computer-generated three-dimensional image holograms". *Applied Optics*, Vol. 31, No. 2, pp.223-229, 1992.

[10] M. Halle, *Multiple Viewpoint Rendering for Three-Dimensional Displays*. Ph.D. Thesis, MIT Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1997.

[11] nVidia Corporation, "NVIDIA Quadro FX Product Overview". 2004.

[12] nVidia Corporation, "NVIDIA Quadro FX 3000G Tech Brief". 2003.

[13] nVidia Corporation, *Cg Toolkit User's Manual*. 2004.

[14] nVidia Corporation, "NVIDIA 'CineFX' Architecture". 2002.

[15] OpenGL Architecture Review Board, D. Shreiner, et al, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4, Fourth Edition*. Addison-Wesley Publishing Company, 2003.

[16] W. Plesniak, *Haptic holography: an early computational plastic*. Ph.D. Thesis, MIT Program in Media Arts and Sciences, Massachusetts Institute of Technology, 2001.

[17] M. Halle, *The Generalized Holographic Stereogram*. M.S. Thesis, MIT Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1991.

[18] M. Lucente, T. Galyean, "Rendering Interactive Holographic Images". *Computer Graphics Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH*, pp. 387-394, 1995.

[19] M. Lucente, "Holographic bandwidth compression using spatial subsampling", *Optical Engineering*, 1996.

[20] S. Venkatasubramanian, "The Graphics Card as a Stream Computer". SIGMOD-DIMACS Workshop on Management and Processing of Data Streams, 2003.

[21] V. M. Bove, J. A. Watlington, "Cheops: A Reconfigurable Data-Flow System for Video Processing". *IEEE Transactions on Circuits and Systems for Video Technology, 5*, Apr. 1995, pp. 140-149, 1995.

[22] J. S. Underkoffler, "Occlusion processing and smooth surface shading for fully computed synthetic holography". *Proc. SPIE Practical Holography XI, 3011*, pp. 19-30, 1997.