2011

3D geometry compression with Holoimage

Nikolaus L. Karpinsky
Iowa State University

3D geometry compression with Holoimage

by

Nikolaus Karpinsky

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Human Computer Interaction; Computer Engineering

Program of Study Committee:
Song Zhang, Co-major Professor
James Oliver, Co-major Professor
Eliot Winer

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Song Zhang for his guidance and support throughout this research and writing. His guidance, enthusiasm, and patience has inspired me; continually renewing my interest and holding me to the highest set of standards. I would also like to thank my committee members for their efforts, guiding and encouraging me to completion: Dr. Jim Oliver and Dr. Eliot Winer. A thank you goes out to my friends and family who put up with me during my research and writing, dropping off the occasional coffee and forcing me to relax when needed. Finally, I want to thank my parents Kevin and Sharon who instilled all of their knowledge onto me, fostering all of my gifts, making this thesis possible. Mom and Dad, I love you.

ABSTRACT

Holoimage is a technique that is capable of compressing 3D geometry scans into 2D images. The main goal of this thesis was to develop a way to compress 3D geometry coming from a structured light scanner into a manageable format. Structured light scanners have made the acquisition and display of 3D data simple. Recently an area of 3D scanning has emerged called realtime 3D scanning, which allows for the capture, reconstruction, and display to be realized in 3D. With these advancements comes the challenge of working with the large volume of data that comes from such a scanner. In an uncompressed format, these realtime 3D scanners can have data rates surpassing 400 MB/s. For realtime applications, this amount is unmanageable, thus a method to compress the data must be found.

Holoimage is a technique that was developed to compress data from such a scanner, converting the raw 3D geometry into 2D images which can then be compressed using 2D image compression techniques. This conversion from 3D to 2D is achieved through a virtual structured light scanner, which is similar to an actual structured light scanner with some key differences. In the virtual system, traditional sources of error such as lighting, camera properties, and system calibration can be controlled to provide an ideal scanning system. Thus, unlike traditional structured light systems, Holoimage does not suffer from disadvantages such as the inability to measure discontinuous surfaces or surfaces with large step height variations. Also, the Holoimage technique is constructed in such a way that all of the steps are pixel operations, thus it can be run on parallel hardware such as a graphics processing unit (GPU).

To further increase the compression of 3D geometry in the Holoimage format, 2D image compression such as portable network graphics (PNG) or joint photographic experts group (JPEG) can be applied. Since the JPEG format is a form of lossy 2D compression, this form of compression introduces error into the reconstructed 3D geometry. Investigations of this error are performed with three different experiments, drawing conclusions from each to construct a structured light pattern that is more resilient to the effects of this lossy compression. In the end, results are shown which allow for compression ratios of over $(72 : 1)$ with root mean squared error of less than 0.1%. If further lossy compression is applied, compression ratios of over $(370 : 1)$ can be achieved with root mean squared error of less than 4.0%. In all this thesis documents previous work in the area of 3D geometry compression, the principle of the Holoimage technique, methods to implement the technique on parallel hardware, experiments on properties of the resulting images, and avenues for future work on the technique.

CHAPTER 1. INTRODUCTION

Advancements in 3D imaging and computational technology have made acquisition and display of 3D data simple. Techniques such as structured light, stereovision, and LIDAR have led the path in 3D data acquisition. Stereoscopic displays have made the display of 3D data a reality. However, as these fields and techniques evolve a growing problem is being confronted; how can 3D data be efficiently stored and transmitted? This thesis describes a novel approach that allows 3D data to be compressed into 2D images and stored in a lossy format. This chapter will present an overview of structured light scanning, explain why compression is needed, discuss the proposed approach, and present the structure of this thesis.

1.1 Overview

Advancements in 3D imaging and computational technology have made acquisition and display of 3D data simple (Zhang, 2010). As the computer graphics field continues to mature, practitioners continue to create new ways to make photorealistic virtual worlds. 3D scanners allow these practitioners to scan real world objects into photorealistic virtual objects giving them an easy way to create art assets. Techniques such as structured light, stereovision, and LIDAR have led the path in 3D data acquisition (Gorthi and Rastogi, 2010). Recently an area of 3D scanning has emerged called realtime 3D scanning which allows for the capture, reconstruction, and redisplay of 3D models in realtime (Zhang, 2010). This allows for a new level of photorealistic virtual worlds and applications that

make use of this realtime technology such as live 3D video or 3D video conferencing. With the advent of this new technology comes new challenges in terms of storage and computing power. A structured light scanner with a resolution of 640×480 yields 307,200 vertices per frame; at 30 frames per second (FPS) this is 9,216,000 vertices a second, which is over 110 MB/sec. This staggering amount of 3D information is too much to store or transmit across a network in realtime.

1.2 Structured Light Scanning

Structured light scanning is a subfield of non contact 3D measurement (Salvi et al., 2010). It is known as active scanning, since a pattern is projected on the object being measured, versus passive which merely observes the object and infers its shape. Structured light scanning works by projecting a known structured pattern onto an object, and then capturing the resulting scene from another angle. The structural pattern assists in finding corresponding points between the projecting perspective and the capturing perspective. If the correspondence between the projecting perspective and capturing perspective is known, and the system is properly calibrated, 3D information can be retrieved (Zhang and Huang, 2006c).

Within structured light scanning a new subfield is emerging, realtime structured light scanning. In this subfield, structured light scanning systems can acquire, reconstruct, and display 3D geometry in realtime. Techniques are emerging which allow for high resolution realtime scanning, making the ability to perform highly detailed scans of objects in realtime a reality (Karpinsky and Zhang, 2010b). With these highly detailed scans comes the challenge of handling all of this data. Conventional 3D model formats store a minimal amount of vertices, and then apply normal maps, bump maps, and textures to achieve high visual fidelity. If high resolution models with animation are stored, the vertices are typically constrained to a few skeletal points which are then

animated. During animation these skeletal points are the ones that get animated, and then the other points move within their constraints of the skeletal points. This allows long series of animation to be performed on just a couple of points yielding small file sizes for large amounts of animation. With video captured from 3D scanners, each point is animated thus each frame of the animation contains the data for the entire model. This yields extremely detailed meshes and animation, but at the cost of high amounts of disk space ocupation.

1.3 Need for Compression

In a structured light scanner, a 3D coordinate is reconstructed for each pixel that is captured, resulting in a large mesh; storing these meshes requires a large amount of space. To illustrate this point consider a simple 2D color image that would come from a movie frame. Each pixel has 8-bit color depth resulting in 24 bits per pixel or 3 bytes per pixel. Now consider a 3D model generated from a structured light scanner, with a 4 byte floating point number for each component of the 3D coordinate (x, y, z). This results in 12 bytes per point, making the 3D geometry an order of 4 times larger. If connectivity information along with point normals are added, which is standard in most 3D formats, the 3D geometry becomes more than 10 times larger. Table 1.1 illustrates this point, where even one of the smallest 3D formats designed for point cloud compression, PLY is still over 5 times larger than a uncompressed 2D image. Employ 2D image compression such as PNG or JPEG compression and the ratio becomes staggering.

1.4 Proposed Approach

Since 3D geometry that is scanned by a structured light scanner comes from 2D images, storing the original 2D images is a natural way to compress the 3D geometry. Then when the 3D geometry is needed, it can be decoded back into 3D. Two major

	Bitmap Image	PLY	DAE	OBJ	STL
File size:	1.2MB	6.5MB	10.6MB	12.8MB	17MB
Ratio:	1 : 1	1 : 5.42	1 : 8.83	1 : 10.67	1 : 14.17

Table 1.1 3D file formats compared to an uncompressed 2D image. All of the formats are 640×480 points. Note the closest format is still over 5 times as large as its 2D counterpart. Also 3D formats contain only vertices and connectivity if required; no point normals or texture coordinates are stored.

limitations exist: the 2D images captured are intolerant to noise and thus cannot be stored in a lossy format; certain structured light systems project many images to capture a single 3D geometry, thus the 2D format can be larger than the 3D format. Using the natural 2D format, there exists a technique of rescanning 3D geometry back into 2D virtually, so that a minimal number of images may be used that can be stored in a lossy format. This technique is entitled Holoimage.

The original intent of Holoimage was to represent 3D geometry with a 2D image (Gu et al., 2006). Holoimage was also used to merge separate scans from a structured light scanner into a single scan with the combined information (Zhang and Yau, 2008). This merging was done by creating a virtual structured light scanner and re-scanning the separate 3D scans. By re-scanning the two separate scans, a single unified scan could be created. Both of these implementations made use of the three-fringe-image technique (Zhang and Huang, 2006a), allowing the scans to be saved in a single image. However, this implementation has limitations due to the required spatial phase unwrapping: it cannot handle large step height changes, nor can it resolve discontinuous surfaces. These limitations existed in the scanner used to gather the original scans, so these limitations were acceptable as the geometry being rescanned would not have large step height changes or discontinuous surfaces.

Currently there are fringe projection techniques which allow for the acquisition of geometry with large step height changes and discontinuous surfaces but require more than three fringe images. Due to these advancements, the limitations of Holoimage

are no longer acceptable, as information is lost, if Holoimage is applied to these new techniques. This thesis presents research on how to make use of different fringe patterns and create a pattern which can be embedded in the red, green, and blue color channels of an image, yet scan geometry with large step height changes and discontinuous surfaces. If this can be done, and the pattern is carefully constructed, a 3D scan can be saved into a single 2D image and stored in a lossy format. This thesis looks at changing the patterns used in Holoimag to mitigate the previous limitations and store 3D geometry in a lossy format.

1.5 Thesis Overview

The following chapters will look into relevant literature, methods and procedures for Holoimage, compression experiments and results, and future work and conclusions. Chapter 2 reviews structured light scanning, fringe projection, phase shifting, and phase unwrapping, along with a literature review of 3D compression techniques which have been proposed for 3D scanning. Chapter 3 gives a detailed discussion on the theory of Holoimage, provides derivations for how to convert from coordinates to phase and from phase to coordinates, along with the discussion on how to implement Holoimage with the OpenGL Shading Language (GLSL). Chapter 4 explores how changing the pattern used in the Holoimage system affects its ability to withstand lossy 2D compression. Finally, chapter 5 will talk about future directions for Holoimage research, specifically how to extend it to video and other color spaces, and then will conclude the findings from previous chapters.

1.6 Summary

Realtime structured light scanning has enabled the acquisition, reconstruction, and display of 3D geometry in realtime. This advancement comes at the cost of large amounts

of data that conventional 3D geometry formats cannot hold effectively. Since the 3D geometry is generated from 2D images, these images are a natural format to use to re-compress the 3D geometry. The technique of Holoimage has been used to merge separate 3D scans into a unified scan, but cannot reconstruct geometry with large step height changes or discontinuous surfaces. If the fringe images used by this technique can be modified, these limitations can be circumvented, and the Holoimage technique can be used to store 3D geometry in a compressed 2D format.

CHAPTER 2. RELATED WORK

Compression of 3D geometry has been increasingly studied, as 3D models are becoming more complex requiring more space. Realtime structured light scanning is a technique which allows these complex models to be generated in realtime, resulting in 3D video. To give some insight into the complexity of 3D scanning, an introduction to structured light scanning is presented along with an overview of phase shifting, phase unwrapping, and an example from a real scanner. Then this chapter will give a short introduction into some relevant methods to encode the data coming from a 3D scanner. A short introduction to each method along with discussing some advantages and disadvantages of each will be presented.

2.1 Structured Light

Before we investigate how to compress 3D geometry coming from a realtime 3D scanner, it is important to understand how 3D geometry is digitized. This section reviews how structured light scanning works along with relevant works in the field. An overview of structured light scanning is presented, followed by fringe projection, phase shifting, and then phase unwrapping. Finally, examples will be presented to illustrate this technique.

2.1.1 Overview of Structured Light Scanning Techniques

Structured light scanning is the process of projecting a known structured pattern onto an object, and then recording from another angle. Figure 2.1 gives a conceptual diagram of this process. Point C in Figure 2.1 illustrates a point being projected, point B is where it lands on the geometry, and point A is where the point is being captured by the camera. The structured pattern that is projected helps find corresponding points ($AandC$) between the projector and camera. This approach of 3D scanning is known as active scanning as a pattern is projected. The other approach to 3D scanning is known as passive, where a scene is simply observed from multiple angles and then corresponding feature points are found through the texture (Salvi et al., 2010). With either approach once corresponding feature points are found (A, B, and C in Figure 2.1), 3D coordinates can be triangulated, assuming that the system is properly calibrated (Zhang and Huang, 2006c).

Since structured light systems project a pattern, they do not need to rely on the natural texture and therefore can find corresponding feature points more accurately and reliably. Typically, a feature point is found for each pixel within the camera allowing for high resolution scans of geometry. Recently, a new classification of structured light scanners has emerged known as realtime structured light scanners. Systems in this classification can acquire, decode, and display geometry in realtime (Zhang and Huang, 2006b). One of the realtime scanning techniques is known as fringe projection. Typically multiple patterns need to be projected and captured to reconstruct a single 3D shape due to noise and required information. By using a fringe projection approach, the number of images projected can be minimized, increasing the temporal resolution to realtime speeds.

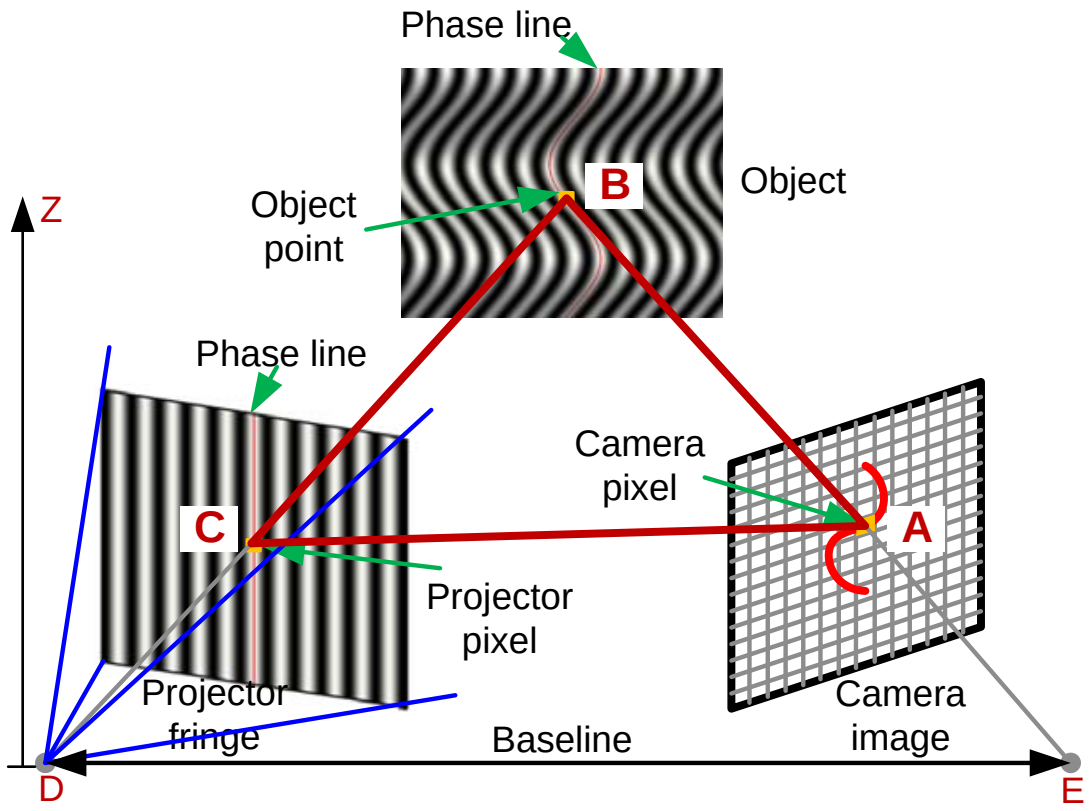


Figure 2.1 Schematic diagram of a structured light system setup. Point A is what the projector is projecting, Point B is where it falls on the geometry, and point C is what the camera sees. If points A, B, and C can be determined the 3D geometry can be triangulated.

2.1.2 Fringe Projection

In structured light scanning two types of patterns are commonly used, binary coded and sinusoidaly varying patterns (Gorthi and Rastogi, 2010). With binary coded patterns multiple different patterns are projected and captured. When the patterns are overlaid on each other, they form a unique coded pattern for each pixel. Since each projector pixel represents a binary value, the resolution of the system is limited to the projects resolution. For higher spatial resolution systems, this approach is undesirable.

Sinusoidaly varying patterns known as fringe patterns increase the spatial resolution beyond the projector's resolution but require phase unwrapping. Instead of using images

with binary grayscale values, fringe projection uses sinusoidally changing intensity values for the structured light being projected. Like binary structured patterns, more fringe images can be used to achieve higher accuracy, but this slows down the measurement speed. To reach realtime 3D imaging, a small number of fringe images (patterns) are typically used. Under certain conditions, only a single fringe pattern is needed to reconstruct the 3D information, thus very high speed may be realized (Takeda and Mutoh, 1983). However, as the complexity of the geometries surface increases, the measurement accuracy is affected, requiring more fringe patterns (Guo and Huang, 2008; Huang and Zhang, 2006).

2.1.3 Phase Shifting

A phase shifting method is usually used to achieve camera pixel-by-pixel spatial resolution during 3D shape recovery. Phase-shifting algorithms are extensively used in optical metrology because of their precision and speed. Over the years, a number of phase-shifting algorithms have been developed including three-step, four-step, least-square algorithms, etc. (Schreiber and Bruning, 2007). All of these algorithms differ in the number of fringe images required and the amount of phase shift, but they all share the same properties: (1) high measurement speed, due to a minimal number of images being projected to recover a 3D shape; (2) high spatial resolution, because the phase can be obtained pixel-by-pixel, thus the measurement can be performed pixel-by-pixel; (3) less sensitivity to surface reflectivity variations, since the calculation of the phase will automatically cancel out the DC components.

In a real world 3D shape measurement system using a fringe projection technique, a three-step phase-shifting algorithm is used extensively in high-speed applications because it requires the least number of fringe patterns for 3D shape recovery. The fringe images of a three-step phase-shifting algorithm with equal phase shift can be described as

$$I_1(x,y) = I'(x,y) + I''(x,y)\cos(\phi - 2\pi/3), \tag{2.1}$$

$$I_2(x,y) = I'(x,y) + I''(x,y)\cos(\phi), \tag{2.2}$$

$$I_3(x,y) = I'(x,y) + I''(x,y)\cos(\phi + 2\pi/3). \tag{2.3}$$

In these equations, $I'(x,y)$ is the average intensity, $I''(x,y)$ the intensity modulation, and $\phi(x,y)$ the phase to be solved for. Figure 2.2 (a) gives a visual representation of these equations. The phase can be obtained by simultaneously solving equations (2.1)-(2.3).

$$\phi(x,y) = \tan^{-1}\left[\sqrt{3}(I_1 - I_3)/(2I_2 - I_1 - I_3)\right]. \tag{2.4}$$

Since the arctangent function only ranges from 0 to 2π, the phase value provided from Equation (2.4) will have 2π phase discontinuities shown in Figure 2.2 (b). To obtain a continuous phase map, a phase unwrapping algorithm is usually needed (Ghiglia and Pritt, 1998), which will be discussed next.

2.1.4 Phase Unwrapping

As stated before, phase unwrapping is used to remove 2π phase jumps that occur during phase shifting. Figure 2.2 (b) gives an example of phase jumps that occur from phase wrapping, Equation (2.4). A phase unwrapping algorithm will traverse along the phase map, adding multiples of 2π to remove these discontinuities. This results in an unwrapped phase map which is given by Equation (2.5), and shown in Figure 2.2 (c).

$$\Phi(x,y) = \phi(x,y) + 2\pi \times K(x,y). \tag{2.5}$$

$\Phi(x,y)$ is the unwrapped phase, $\phi(x,y)$ is the wrapped phase obtained from Equation (2.4), and $K(x,y)$ is the integer number to represent phase jumps. The key to a phase unwrapping algorithm is quickly and correctly finding $K(x,y)$ for each pixel in

the phase map. All spatial phase unwrapping algorithms have two common limitations: they cannot resolve large step height changes that cause phase changes large than π, nor can they resolve discontinuous surfaces.

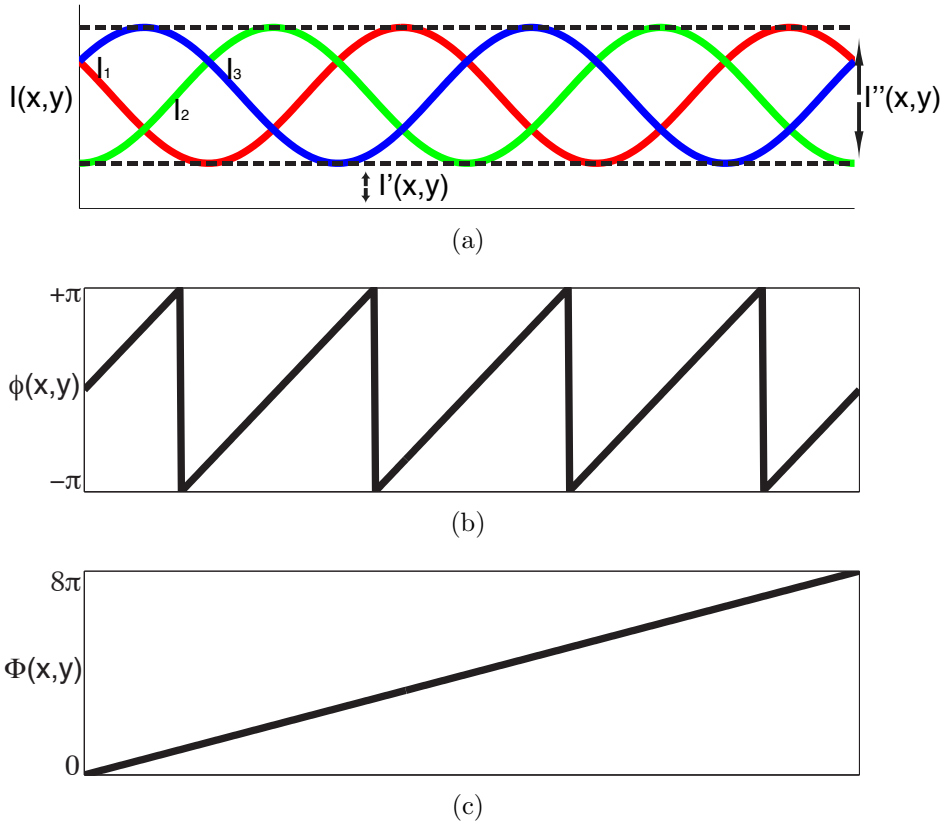
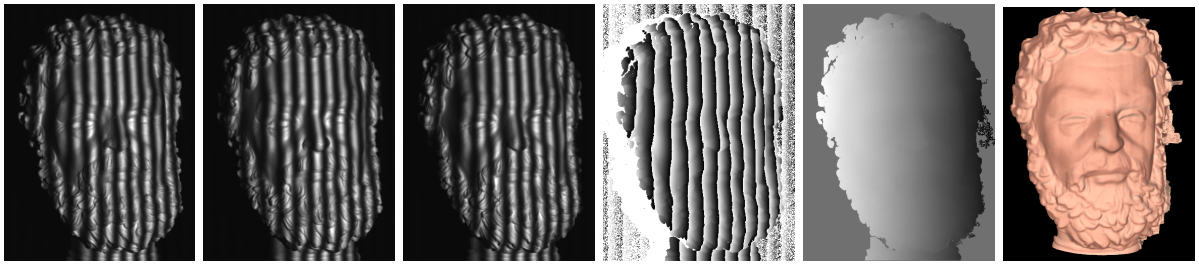


Figure 2.2 Example of three-step phase-shifting algorithm. (a) shows the fringe images I_1, I_2, and I_3, (b) contains the wrapped phase $\phi(x, y)$, and (c) contains the unwrapped phase $\Phi(x, y)$.

2.1.5 3D Shape Measurement Results with the Phase-Shifting Technique

To better illustrate how structured light scanning with fringe projection works an example is presented. The three fringe images modeled by Equations (2.1)-(2.3) are projected onto the object being scanned, and then captured from a camera at another angle. Figure 2.3 (a)-(c) shows the captured fringe images, and shows that the geometry has distorted the fringe pattern. Applying Equation (2.4) to each pixel will result in

a wrapped phase map as shown in Figure 2.3 (d). In the wrapped phase map the 2π discontinuities can be visualized by the color changing from white to black. Next a phase unwrapping algorithm is applied and the unwrapped phase map is generated, which is shown in Figure 2.3 (e). If the system is calibrated, then 3D geometry can be reconstructed. Figure 2.3 (f) shows the final recovered 3D shape.



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

Figure 2.3 Example of a fringe projection scanner digitizing 3D geometry. (a)-(c) shows the fringe images that are projected onto the geometry, model by Equations (2.1)-(2.3), (d) shows the wrapped phase map which is generated by applying Equation (2.4) to (a)-(c). (e) shows the unwrapped phase map which is generated by applying Equation (2.5) to (d). Finally (f) is the reconstructed geometry that is retrieved from (e).

By taking this scanning process and speeding it up to realtime speeds, a realtime structured light scanner is created. With such a realtime scanning system, the challenge of storing, transmitting, and displaying this data emerges. The scan showed in Figure 2.3 (f) consists of 640×480 points; in the OBJ file format this results in a 16.4MB file. Thus, compression methods need to be explored, so that the data can be put into a manageable format.

2.2 Overview of 3D Compression Techniques

Compressing 3D data is a challenge that has been studied for the last decade. Many different methods for compressing or representing 3D data have been proposed, each

with its own advantages and disadvantages. For the sake of brevity, only a few key methods that are closely related to this proposed method within this large body of literature are examined, namely, depth mapping, multiview depth mapping, geometry images, heuristic based point cloud compression, and level of detail meshes.

2.2.1 Depth Mapping

To overcome the challenge of compressing large static models, computer graphics has employed what is know as depth mapping for sometime. The idea behind depth mapping is to encode 3D geometry into 2D images, which can then later be decoded back to 3D, known as image based rendering (Krishnamurthy et al., 2002; Chai et al., 2004). Typically, the model being depth mapped is aligned with a plane such as the xy plane, and then the z component is encoded with a 2D image known as a depth map. The result of the process is a 2D image, which assumes that it is xy axis aligned, the points are uniformly spaced, and each pixel encodes the depth at the point or the z value. Performing this operation allows for the use of decades of existing research in 2D image processing to be leveraged as the 3D geometry has been encoded into a 2D image. Thus, storage and transmission of the geometry is simplified. Typically, large static models in 3D worlds are terrain models, and depth mapping is often employed to quickly generate these models at photorealistic levels. A problem with this method is that coordinates in models are not always uniformly spaced, which is a requirement of the depth map, thus geometry needs to be resampled to be put into the depth map. Another limitation is that depth maps are typically saved to an 8-bit image, thus the depth resolution is limited to 256 values. Due to nature of 3D scanners and their use of 2D images to generate 3D data, a natural connection with depth mapping is apparent. In a typical 3D scanning system, the 3D coordinates are recovered from 2D images captured by cameras. Therefore, there should be a way to convert these recovered 3D scenes back into 2D images.

2.2.2 Multiview Depth Mapping

As 3D video and 3DTV becomes more mainstream, standards are emerging for video plus depth. The technique of depth mapping along with video works nicely for a single view reconstruction, as left and right eye images can be constructed on the fly at the receiver side before display (Kauff et al., 2007). As autostereoscopic displays emerge with multiple view points, this approach no longer works well. Instead N different video plus depth streams are required based on the number of views to be presented. Since this increases the bit rate of the stream by a factor of N, this is only acceptable for a small N. To cope with this, techniques are trying to make use of redundant information within the depth maps, then encoding only the changing depth values, but this requires a time consuming encoding procedure. Currently the motion picture experts group (MPEG) is working on the specifications for an auxiliary video data representation for video plus depth.

2.2.3 Geometry Images

Geometry images developed by Gu et al. (2002) is another technique, which compresses 3D geometry into 2D images. The basic premise behind the technique is to find a way to cut a model and then unwrap it into a flat disc. Each point in the disc corresponds to a single geometry point. Coordinates (x, y, z) for each point on the disc are then encoded into an image as red, green, and blue (RGB). This allows 3D geometry to be projected onto a 2D image and then brought back to 3D. The constraints of this technique are that geometry images cannot represent non-manifold geometry, and greater distortion and less uniform sampling can occur when unwrapping an entire mesh as a single chart.

Another disadvantage of geometry images is the encoding process. Since the model needs to be cut in order to unwrap it into a flat disc, appropriate cut locations must be

found. This is performed through an iterative cutting algorithm which weights various cuts, finding the minimal cuts needed to unwrap the geometry. As the genus of the geometry increases though, more cuts are needed, which takes long periods of time to find. Even with low genus models, less than six, and a face count of approximately 300K it takes about 1 hour to convert. This makes realtime encoding challenging, thus geometry images are not applicable for the transmission of data from a realtime structured light scanner.

2.2.4 Heuristic Based Point Cloud Compression

Heuristic based point cloud compression is a lossless method of compressing point cloud data from a 3D scanner. The idea behind this technique is to start at a single point, then move to a neighboring point based off a set or rules, and then encode a correction vector. Different methods use different heuristics to try and guess the next neighboring point (Merry et al., 2006; Gumhold et al., 2005). When decoding the point cloud, the decoder starts at the first point, then predicts the location of the second point and corrects with the encoded correction vector. If the prediction heuristics are good, the correction vectors become very small, yielding low bit rates and thus high compression of the point clouds.

Challenges of these approaches include picking correct heuristics, achieving a high speed on large meshes, and storing connectivity information if it is present. As stated previously, the choice in heuristics determine what the resulting file size is. If they are poorly chosen, the file size could result in a similar size as the uncompressed mesh. Achieving high speed is also difficult with these methods as the operations are typically serial operations. Gumhold et al. (2005) were able to achieve an encoding speed of 20 seconds per mesh on a 100K point mesh. For high resolution real time scanning this is not viable since the frame rate would be too slow. Finally connectivity information cannot be easily stored since the neighboring points are selected based on a heuristic.

If this information would be stored it would have to be stored separately in a different compressed medium and applied after the mesh is decoded from its encoded state.

2.2.5 Level of Detail Meshes

A technique that is often employed with depth mapping and geometry images is level of detail (LOD) meshes (Lindstrom et al., 1996). As a camera moves farther from its subject, the perspective makes the subject appear to get smaller and smaller. As the subject gets smaller from the cameras viewpoint, less camera pixels can capture the subject, thus the subjects level of detail can be decreased without affecting the detail at the cameras viewpoint. LOD meshes take advantage of this principle and reduce the number of vertices in the overall mesh to display an appropriate level of detail, while displaying as few vertices as possible for speed. Since depth maps are applied per vertex, the fewer vertices the faster the decoding process. Thus LOD meshes work nicely with depth mapping, decoding and displaying only the needed level of detail in a mesh. The challenge in LOD meshes is how to determine which vertices to drop in a fast and efficient manor. Since this technique is supposed to aid in speeding up rendering, it needs to be extremely fast and efficient.

2.3 Summary

Over the past decade there have been many ways that have been developed to compress 3D point clouds and 3D scans. Examples of these methods include depth mapping, multiview depth mapping, geometry images, and heuristic based point cloud compression. Furthermore, level of detail meshes have helped speed up decoding with these methods. Each of these methods come with advantages and disadvantages. In addition to new compression methods, new methods for acquiring 3D geometry have been discovered, namely structured light scanning. With structured light scanning and fringe

projection, a special classification of systems has emerged entitled, realtime structured light scanners. These systems are able to acquired, decode, and display in realtime, making 3D video a reality. This chapter examined the basic principles behind structured light scanning, specifically fringe projection, phase shifting, and phase unwrapping. An example with real data was shown, providing an illustration of how structured light scanning with fringe projection works.

CHAPTER 3. METHODS AND PROCEDURES

To compress 3D scanned data, the technique of Holoimage is proposed. This technique effectively depth maps a 3D scene through the use of a virtual fringe projection scanner. The fringe pattern is constructed so that all fringe fit in the red, green, and blue (RGB) color channels of a 2D image, and can be calculated in parallel. Parallel calculation allows this technique to be performed on a graphics processing unit (GPU) through the use of the OpenGL Shading Language (GLSL). This chapter gives an introduction to the principle, how encoding is performed, how encoding through GLSL is performed, how to decoding a Holoimage, how to decoding though GLSL, and finally presents some experimental results to verify the performance of this proposed technique.

3.1 Holoimage

The proposed Holoimage technique uses a virtual fringe projection system achieved through the use of OpenGL Shading Language (GLSL) Shaders. The pattern that is chosen is a composite phase shifting pattern which allows for the phase to be solved for pixel by pixel (Karpinsky and Zhang, 2010a). This section gives the principle behind the Holoimage technique, how encoding is performed, how encoding can be implemented with GLSL Shaders, how decoding is performed, how decoding can be implemented with GLSL Shaders, and finally gives some experimental results.

3.1.1 Principle

The principle behind Holoimage (Gu et al., 2006) is borrowed from optical metrology and is known as fringe projection explained in Section 2.1.2. Figure 3.1 shows the Holoimage system schematic, which consists of a projector and a camera. The projector projects a structured pattern or structured light onto an object, and a camera captures the resulting scene. As the structured pattern from the projector lands on the objects in the scene, the 3D geometry distorts the pattern, which is what the camera captures. Assuming that the geometric relationship between the projector pixels and the camera pixels are known, the 3D geometry can be reconstructed from the distortion between each image (Zhang and Huang, 2006c). Thus 3D geometry is transformed into a single 2D image, and then the 2D image can be used to reconstruct the 3D geometry.

In the Holoimage setup, Figure 3.1, the system differs slightly from a real 3D fringe projection system, Figure 2.1, in that the camera and projector are both virtual orthogonal devices instead of perspective ones. In a real system the pinhole camera model, a perspective projection, is used which complicates the technique of encoding and decoding. The camera and projector lens distortion usually brings 3D shape measurement errors. Thus, using an ideal orthogonal projection simplifies the process further. Another difference is that in a real 3D fringe projection system, the light usually cannot pass through an opaque object, but in a virtual fringe projection (Holoimage) system, the fringe patterns can pass through any object to generate fringe patterns for 3D reconstruction. Moreover, since the position of the virtual camera and projector can be precisely configured, the geometric relationship between the two can be precisely defined resulting in no need to calibrate the camera and projector. This is usually a complicated calibration procedure with a real 3D fringe projection system (Zhang and Huang, 2006c). With the Holoimage setup, 3D shape reconstruction is significantly simplified and is highly precise, resulting in a quick and efficient way to depth map a 3D scene.

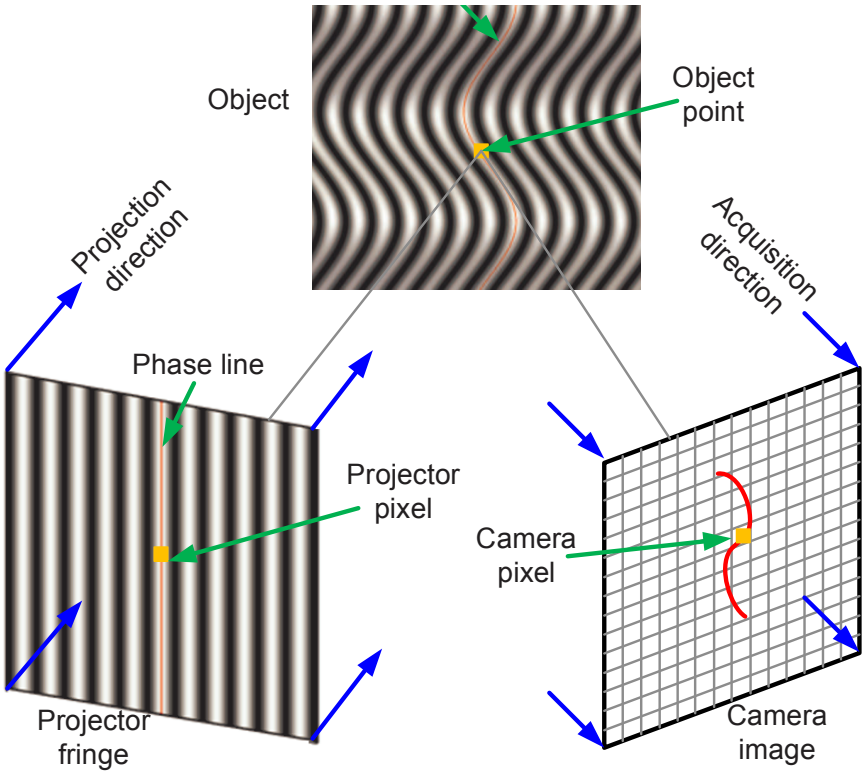


Figure 3.1 Holoimage setup. The projector pixel is what the projector is projecting, into the object, and is captured by the camera at the camera pixel. The projector pixel and object point are achieved through GLSL Shaders, and the camera pixel is the pixel location in the resulting Holoimage.

3.1.2 Encoding - Coordinate to Phase Conversion

Typically, three phase-shifted fringe patterns are required in order to perform 3D shape measurement due to background lighting and random noise. In a virtual Holoimage system, all environmental variables can be precisely controlled; therefore only two phase-shifted fringe patterns are needed to solve for the phase. These two fringe patterns can be modeled and encoded into two of the primary color channels of the projector. Since the background light can be precisely controlled, the fringe images can be ideally sinusoidal and described in the following two equations:

$$I_r(x,y) = [1.0 + \sin(\omega \times x)] \times 0.5, \tag{3.1}$$

$$I_g(x,y) = [1.0 + \cos(\omega \times x)] \times 0.5. \tag{3.2}$$

Where $\omega = 2\pi \times f$, f is the desired frequency of the fringe, I_r is the color assigned to the red color channel, and I_g is the color assigned to the blue color channel. x is the coordinate value ranging from $(0, 1)$ from the projectors perspective. From these two equations, the wrapped phase may be obtained point-by-point.

$$\phi(x,y) = \arctan\left[\frac{I_r(x,y) - 0.5}{I_g(x,y) - 0.5}\right]. \tag{3.3}$$

Similar to the three-step phase-shifting algorithm discussed in Section 2.1.3, this yields a phase value for each pixel that ranges from $[-\pi, +\pi)$ with 2π phase jumps, which can later be used to reconstruct the 3D geometry. The unwrapped phase $\Phi(x,y)$ can be obtained by adopting a conventional spatial phase unwrapping algorithm to find $K(x,y)$. If such an algorithm is used, the technique would not be able to encode large step height variations or discontinuous surfaces. However, since there are three primary color channels and the blue channel is not yet utilized in the Holoimage system, we can encode $K(x,y)$, the integer number to represent phase jumps, into the third channel by

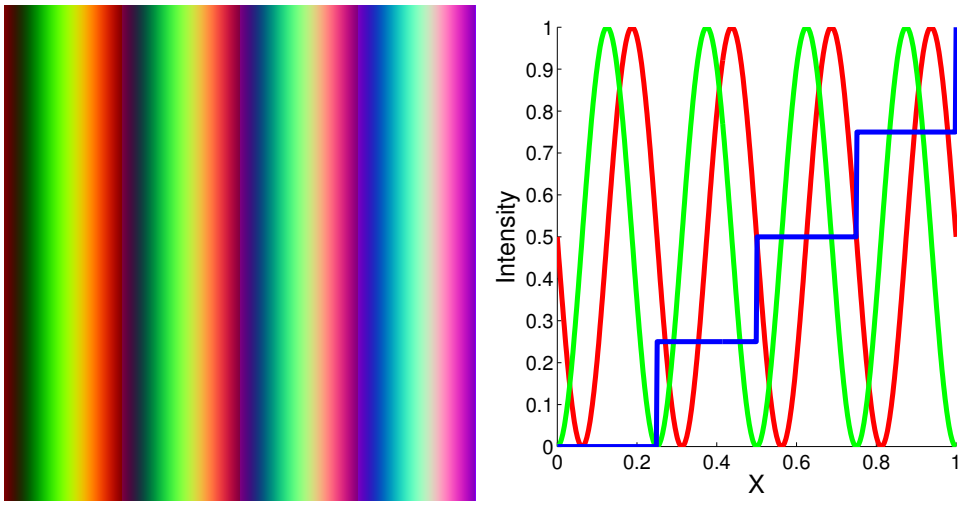
projecting it along with the fringe patterns, given with Equation (3.4).

$$I_b(x, y) = \frac{floor(x \times f)}{f}. \tag{3.4}$$

Here, f is the frequency of the fringe used in Equations (3.1)-(3.2) and x is the coordinate value ranging from $(0, 1)$ from the projectors perspective. To solve for the unwrapped phase $\Phi(x, y)$, the result from Equation (3.3) can be combined with $I_b(x, y)$ to yield an absolute phase, given by Equation (3.5).

$$\Phi(x, y) = \phi(x, y) + I_b(x, y) \times 2\pi. \tag{3.5}$$

Embedding the three color channel functions (I_r, I_g, I_b) in the red, green and blue color channels, a gradient image to be projected is created; this image is seen in Figure 3.2 (a). Figure 3.2 (b) shows a single cross section of the pattern.



(a) (b)

Figure 3.2 Ideal fringe for the Holoimage technique. (a) shows the Holoimage Equations (3.1)-(3.4) displayed as an image, (b) shows a single cross section plot of the three color channels (I_r, I_g, I_b). Note a f of 4 was used to generate this fringe.

Given that there are only three images, these functions can be encoded into the three primary color channels (red, green, blue, or RGB) of a 2D image and projected in the

virtual system at once, achieving depth mapping of 3D geometry into a 2D image. Since the 3D information can be encoded into a single color image, it drastically reduces the size of storing 3D geometry data. In addition, because the phase at each point can be solved for point-by-point without referring to any neighboring point, the decoding can be achieved in parallel. With a highly parallel operation device, such as GPU, the decoding step can be realized quickly.

3.1.3 GLSL Encoding

Since each pixel of the resulting Holoimage is calculated with f which is constant for each Holoimage, and x which is unique for each pixel horizontally, the step of encoding can be performed in parallel. To perform the parallel calculation of the Holoimage, GLSL Shaders can be used to color a 3D scene with the structured light pattern, with small modifications to the rendering pipeline. Once encoded, the scene can be rendered to a texture, retrieved from the GPU via direct memory access (DMA), and saved to an image. Figure 3.3 illustrates the encoding pipeline. The process starts by setting the frame buffer object to render to a new texture. Next the Holoimage encoding shader, detailed in Section 3.1.3.2, is bound and the geometry to be encoded is rendered. Finally the Holoimage encoding shader is unbound, and the texture which the scene was rendered to is saved out as the 2D Holoimage.

3.1.3.1 Model View Matrix

In the OpenGL pipeline before Version 3.2, the model view matrix for a camera is created and passed into the shaders. In versions after 3.2, this is left to the users, but can be applied in the same manner as versions before 3.2. The model view creates the transform that brings coordinates into the camera's perspective before it is projected onto the screen with the projection matrix. This camera perspective is the same camera that is needed in the fringe projection system. In order to color the scene, x is needed

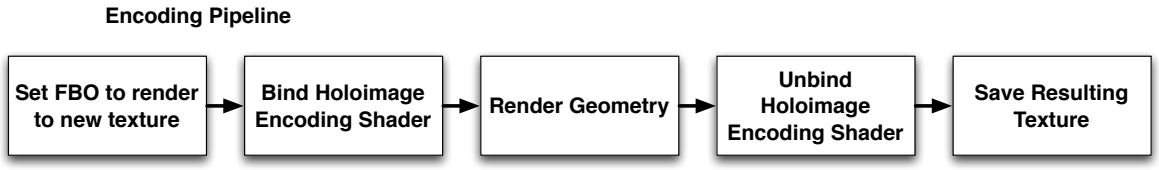


Figure 3.3 Holoimage encoding pipeline. The pipeline starts by setting the frame buffer object (FBO) to render to a new texture. Next the Holoimage encoding shader is bound and then the geometry to be encoded is rendered. Finally the Holoimage encoding shader is unbound, and the texture which the scene was rendered to is saved out as the Holoimage.

from the projector's perspective. To do this a second model view matrix is needed for the projector's perspective. This can be created by acquiring the base model view matrix before any transforms are applied, saving it to the projector's model view matrix, and then rotating the model view matrix some angle θ about an axis (eg. 30° about the y axis in the case of this thesis). Now before the scene is drawn, the Holoimage shader needs to be bound, and any transform that is applied to the cameras model view matrix needs to be applied to the projectors model view matrix as well. When geometry is drawn, the value of the projector's model view matrix must be passed to the shader, so that the appropriate x coordinate value can be determined for each vertex drawn. From this point, the rest of the encoding is performed on the Holoimage shader.

3.1.3.2 Holoimage Encoding Shader

Figure 3.4 gives a graphical representation of how the Holoimage encoding shader works. Since the Vertex Shader is only called once for each vertex, it is not desirable to encode the Holoimage in this stage of the pipeline. If there is a vertex for each pixel it could be done, but the Fragment Shader already gets called for each pixel, thus it is the location to perform the calculation. The Vertex Shader does need to pass the x

position relative to the projector to the Fragment Shader, thus some work does need to be done. To pass x, the Vertex Shader needs to take the projector's model view matrix and multiply it by the vertex and pass it to the Fragment Shader. Figure 3.4 illustrates the Holoimage encoding shader with the Vertex Shader taking in the projector model view matrix and camera model view matrix, and emitting a projector vertex and a camera vertex. This will give the Fragment Shader the x value of the pixel interpolated across the scene.

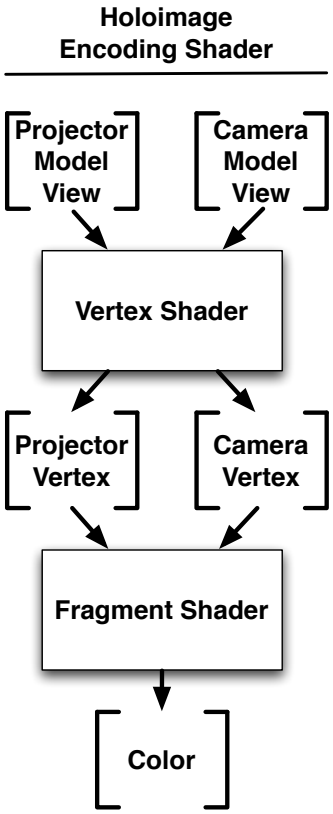


Figure 3.4 Holoimage encoding shader. The Vertex Shader takes in the projector model view matrix and camera model view matrix, and emits both a projector vertex and camera vertex. The Fragment Shader takes both of these vertices and then emits a colored fragment which holds the value of the fringe at the specified fragment.

Within the Fragment Shader is where the actual encoding is performed. The Fragment Shader requires x from the Vertex Shader, and f which is a uniform variable passed from OpenGL CPU side. With these inputs Equations (3.1)-(3.2) and (3.4), are used to color the fragment that is emitted. Figure 3.4 shows this with the Fragment Shader taking in the projector vertex and camera vertex, and emitting the encoded color. Once the scene is encoded with the Holoimage texture, it can be rendered to a texture, retrieved from the GPU, and saved to an image.

3.1.4 Decoding - Phase to Coordinate Conversion

Decoding the resulting Holoimage requires more steps than encoding, but can be efficiently done through triangulation in parallel. Figure 3.5 shows how a single pixel can be decoded once the absolute phase map is acquired. Equation (3.5) gives the unwrapped phase map using the Holoimage. This unwrapped phase map gives a unique phase value for each pixel varying across the x axis.

Decoding a Holoimage is achieved through a very simple triangulation. Figure 3.5 shows the diagram of decoding z from phase for the Holoimage system. It decodes a single depth value z using a reference plane (a flat surface with $z = 0$). In other words, the depth z value is relative to the flat plane. The ultimate goal is to be able to calculate the z value for each point in a point-by-point manner from the computed phase value in Equation (3.5).

To begin, from Figure 3.5, we can use basic trigonometry to find z in terms of $\Delta_X C - A$ and $\tan\theta$, where θ is the angle between the capture plane and the projection plane.

$$z = \frac{\Delta X_{C-A}}{\tan\theta}. \tag{3.6}$$

To simplify the 3D rendering, the graphics pipeline is usually set up in a way that the

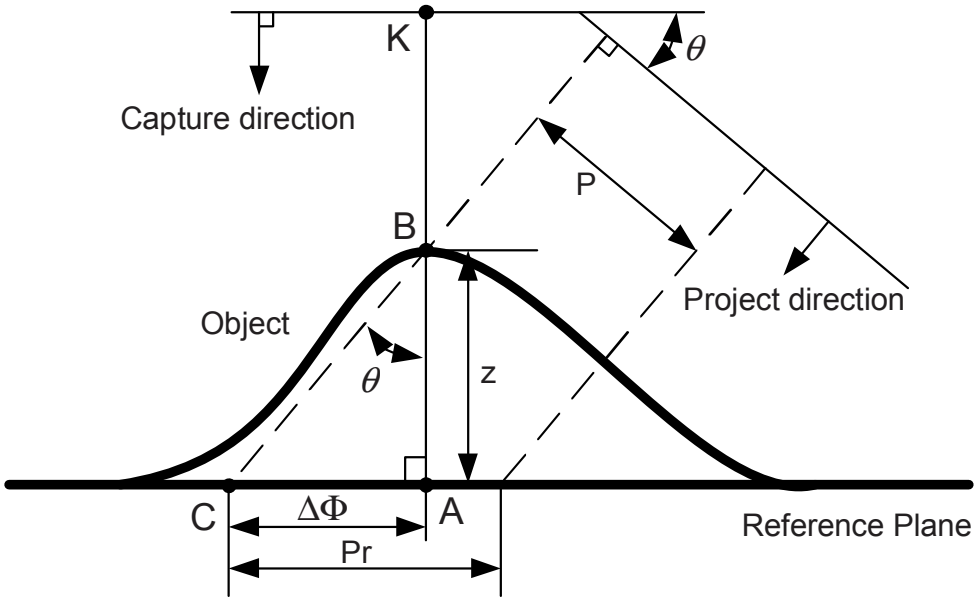


Figure 3.5 Diagram for the decoding of a single depth value z using a reference plane (flat surface with $z = 0$).

rendered scene gets visualized within a unit cube, thus the size of a pixel is $\frac{1}{W}$, where W is the total number of pixels horizontally in the unit cube. If the origin of the coordinate system for the unit cube is aligned with the origin of the image then x can be found by simple scaling, that is

$$x = \frac{i}{W}. \tag{3.7}$$

where i is the index of the pixel being decoded in the Holoimage. Therefore, the distance between C and A in the unit cube is actually:

$$\Delta X_{C-A} = \frac{\Delta i_{C-A}}{W}. \tag{3.8}$$

At this point Equations (3.6)-(3.8) can be combined yielding

$$z = \frac{\Delta i_{C-A}}{W \tan\theta}. \tag{3.9}$$

This gives z in terms of the change of index from point C to point A, along with the number of pixels horizontally, and the angle between the projection and capture planes. Since there is no easy way to find i for point C because it only exists virtually, we will have to look further to see if the phase value can be leveraged.

For an arbitrary pixel K in the Holoimage system, the point A on the reference plane would have a phase value of Φ^r_A. From the camera perspective or the Holoimage perspective, point B would be in the place of point A and the phase value would be Φ_B or just Φ. From the projectors perspective, point B and point C (on the reference plane) have the same value, i.e. $\Phi = \Phi_B = \Phi^r_C$. Since the fringe stripes are uniformly distributed on the reference plane we have the following equation.

$$\Delta\Phi = \Phi^r_C - \Phi^r_A = \Phi_B - \Phi^r_A. \tag{3.10}$$

The phase of a point on the reference plane can be defined as a function of the index i and the fringe pitch (number of pixels per period of fringe) on the reference plane.

$$\Phi^r = \frac{2\pi i}{P_r}. \tag{3.11}$$

Here P_r is the fringe pitch on the reference plane. Again using more trigonometry we can define the fringe pitch on the reference plane in terms of the fringe pitch of the projector.

$$P_r = \frac{P}{\cos\theta}. \tag{3.12}$$

Here, P is the fringe pitch, the number of pixels per period that the projector actually projects, which is simply $\frac{1}{f}$. Combining Equation (3.11) and Equation (3.12), we obtain

the phase of a point on the reference plane in terms of the fringe pitch P and the angle between the capture plane and projection plane θ.

$$\Phi^r = \frac{2\pi i \cos\theta}{P}. \tag{3.13}$$

Furthermore, Equation (3.10) and Equation (3.13) can be combined to obtain

$$\Delta\Phi = \frac{2\pi i_C \cos\theta}{P} - \frac{2\pi i_A \cos\theta}{P} = \Phi - \frac{2\pi i_A \cos\theta}{P}. \tag{3.14}$$

or in another means as,

$$\Delta\Phi = \Phi - \frac{2\pi i_A \cos\theta}{P} = \frac{2\pi \cos\theta \Delta i_{C-A}}{P}. \tag{3.15}$$

Rearranging Equation (3.15) yields

$$\delta i_{C-A} = \frac{\Delta\Phi P}{2\pi \cos\theta}. \tag{3.16}$$

From here we can go back to where we left off with Equation (3.9) and substitute in Equation (3.16).

$$z = \frac{\Delta\Phi P}{2\pi W \cos\theta \tan\theta}. \tag{3.17}$$

Or

$$z = \frac{\Delta\Phi P}{2\pi W \sin\theta}. \tag{3.18}$$

Substituting in $\Delta\Phi$ from Equation (3.15) we obtain:

$$z = \frac{P[\Phi - \frac{2\pi i_A \cos\theta}{P}]}{2\pi W \sin\theta}. \tag{3.19}$$

Now we relate the depth information z with the projected fringe patterns, the Holoimage pixel index, and the setup of the Holoimaging system, that is

$$z = \frac{P\Phi - 2\pi i_A \cos\theta}{2\pi W \sin\theta}. \tag{3.20}$$

This yields a value z in terms of P, the fringe pitch; i_A, the index of the pixel being decoded in the Holoimage; θ, the angle between the capture plane and the projection plane; Φ, the phase at the current pixel being decode in the Holoimage; and W, the number of pixels horizontally.

Because the system is an orthogonal system and the rendering is performed within a unit cube, the x and y coordinates can be calculated by scaling the i and j as,

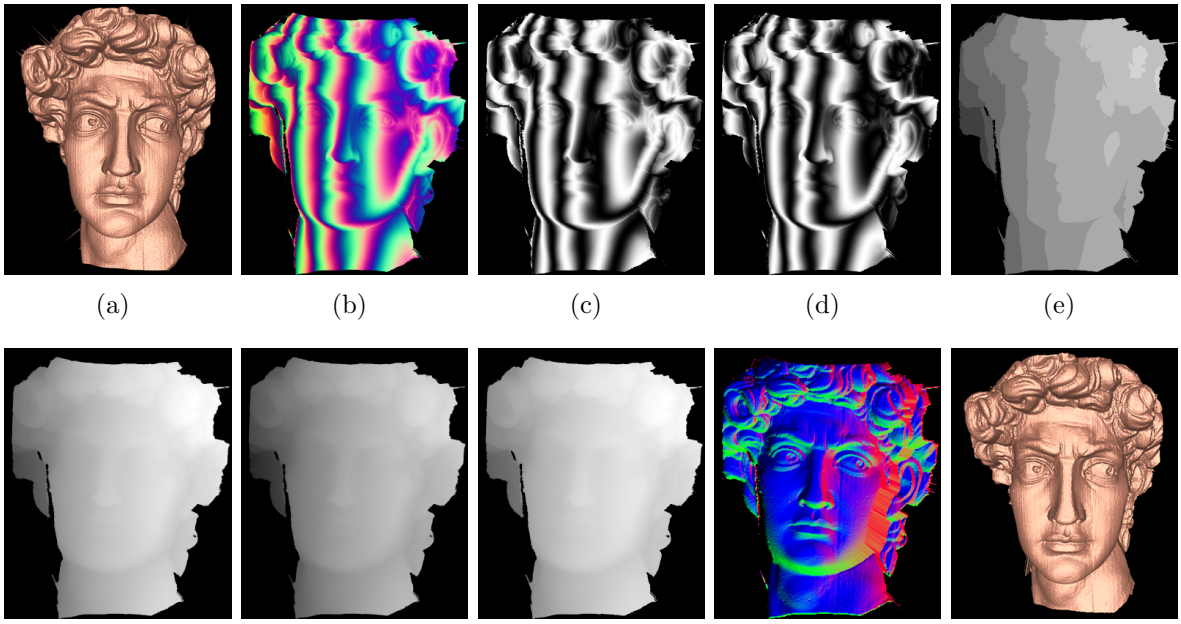
$$x = \frac{i}{W}, \tag{3.21}$$

$$y = \frac{j}{W}. \tag{3.22}$$

All of these terms are specific to the point at which the Holoimage is being decoded, thus making the decoding a point-by-point function; given parallel hardware, a Holoimage scene can be decoded in parallel giving a large speed boost. Figure 3.6 shows the process of converting a Holoimage back to 3D geometry. (a) shows the original geometry, (b) shows the Holoimage after encoding (resolution of 512×512 with $f = 6$, and $\theta = 30°$), (c) - (e) shows the fringe images for the red green and blue color channels, (f) shows the unwrapped phase map, (g) shows the median filtered unwrapped phase, (h) shows the coordinate map (i) shows the normal map, (j) shows the resulting geometry.

3.1.5 GLSL Decoding

For each point the z is calculated with P the fringe pitch, i_A the index of the pixel being decoded in the Holoimage, θ the angle between the capture plane and the projection plane, Φ the phase at the current pixel being decode in the Holoimage, and W the number of pixels horizontally it can be calculated in parallel. Since decoding has



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

(f) (g) (h) (i) (j)

Figure 3.6 Example of 3D geometry encoding and decoding using Holoimage. (a) original scan of David, (b) Holoimage of scan, (c)-(e) fringe images in the red, green, and blue color channels, respectively, (f) unwrapped phase, (g) median filtered phase map, (h) coordinate map (i) normal map, (j) reconstructed figure.

multiple steps unlike encoding, a set of GLSL Shaders are required along with multipass rendering. This allows image processing to be performed on the phase map, creation of a normal map, and then the final rendering. Figure 3.7 illustrates the decoding pipeline.

The start of the decoding pipeline involves creating a texture from the Holoimage and setting the frame buffer object (FBO) to render to a new texture. This setup allows image processing to be done using the GPU, by running the Holoimage through the Fragment Shader and then outputting the result to the texture that is bound by the FBO. The first step in the pipeline is to calculate the phase from the Holoimage. This is done by binding the phase calculator shader and rendering out a screen aligned quad, which applyies Equation (3.5) to each pixel in the Holoimage. Next the FBO is set to a new texture and median filtering is performed on the phase map by binding the median filter shader and rendering out a screen aligned quad. Afterwards, the FBO is again set to another new texture, the coordinate calculator shader is bound, and the screen aligned quad is rendered yielding a coordinate depth map from the filtered phase map. If normals need to be calculated, this can be done by setting the FBO to another new texture, binding the normal map shader, and rendering the screen aligned quad. If a normal map is specified, the previous step can be skipped, and the pipeline can go to final rendering. For final rendering, the FBO is set to the original buffer, either the back or front buffer, the final render shader is bound, and then a plane of pixels is rendered out. The following sections discuss in detail the steps of the pipeline from Figure 3.7. Specifically, we will cover phase calculation, median filtering of the phase, coordinate calculation, normal calculation, and final rendering.

3.1.5.1 Phase Calculator

Before the phase can be calculated the frame buffer object (FBO) must be set to a render texture. This allows the phase calculator shader to render the phase value out to a texture which can be used by shaders later in the pipeline. The texture that it renders

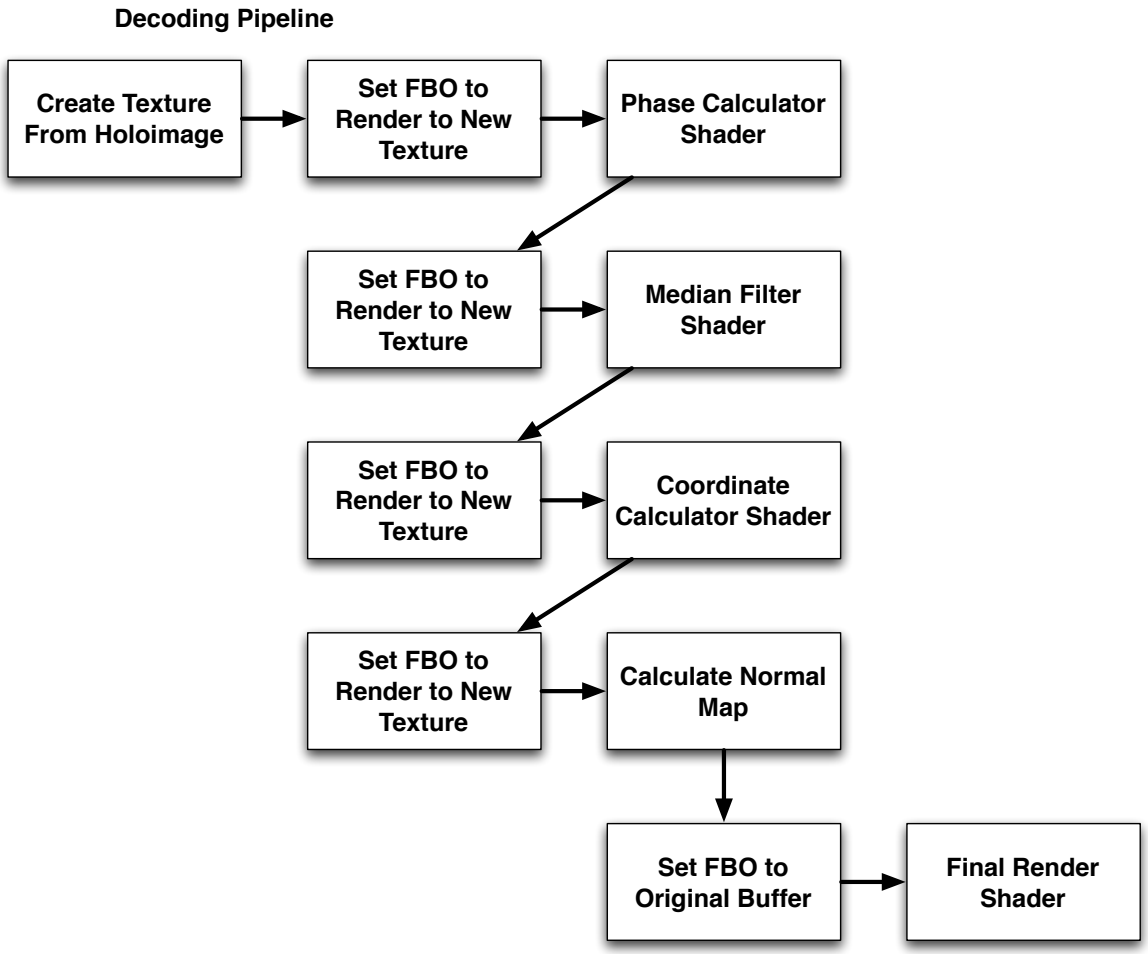


Figure 3.7 Holoimage decoding pipeline. The pipeline starts with creating a texture from the Holoimage to be decoded. Next the FBO is set to render to a new texture and the phase calculator shader is run by binding it and rendering a screen aligned quad. This process is then repeated with the median filter shader, coordinate calculator shader, and normal map shader, each time setting the FBO to render to a new texture. Finally the FBO is reset to the original buffer and the final render shader is run. This takes a plane of pixels and recreates the geometry, rendering the result to the screen.

out to needs to be a floating point texture, and can be single channel or known as a luminance map. Once the FBO is bound, and the phase calculator shader is bound, a screen aligned quad is rendered so that the Fragment Shader is called for each pixel in the Holoimage.

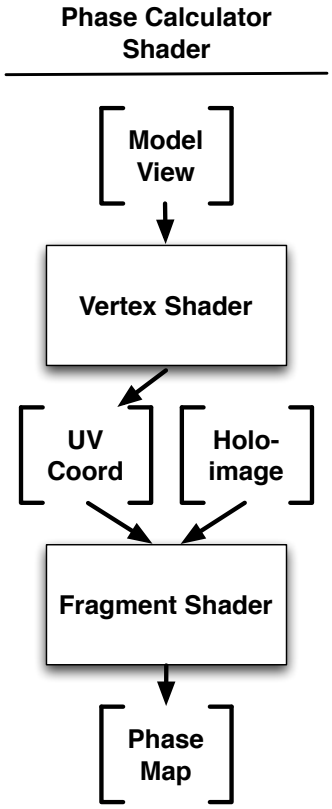


Figure 3.8 Holoimage phase calculator shader. The Vertex Shader takes in the model view matrix and then emits a UV coordinate. The Fragment Shader takes the UV coordinate along with the Holoimage texture, applies Equation (3.5) and emits the phase map.

Figure 3.8 shows how the phase calculator shader works. The phase calculator works by applying Equation (3.5) to each pixel in the Holoimage. The Vertex Shader simply applies the model view projection transform on the vertices, and varies the texture coordinate for the Fragment Shader. Actual phase calculation is done on the Fragment

Shader since it is called for each pixel. In the Fragment Shader, the Holoimage texture is read in, and Equation (3.5) is applied, outputting the result as the fragment color. The Fragment Shader takes in the UV coordinate and Holoimage, and emits the phase map. This effectively calculates the phase for each pixel in the Holoimage and then saves the result in a texture to be used by shaders later in the pipeline.

3.1.5.2 Median Filtering

For median filtering, the same FBO is used with screen aligned quad being rendered, but a different texture output is used. This allows the median filtering to read the phase, filter, and then output to a new texture. Again the texture that it renders out to needs to be a floating point texture, and should be the same number of channels as the previous phase map texture. Once the FBO is bound and the median filter shaders is bound, the screen aligned quad is rendered so that the Fragment Shader is called for each pixel in the phase map.

Figure 3.9 shows how the median filter shader works. The 3×3 median filtering shader is borrowed from McGuire (2008). Once again the Vertex Shader simply applies the model view projection matrix to the vertex, and varies the texture coordinate. Actual median filtering is done in the Fragment Shader. In the Fragment Shader, the neighboring values for a pixel in the phase map are read into an array and a series of dropping the minimum and maximum is performed. The resulting value, once the iterations are done, is the median value and it outputs as the fragment color, being saved in the texture bound by the FBO.

3.1.5.3 Coordinate Calculation

Coordinate calculation once again makes use of the same FBO, screen aligned quad, and a different texture for the output. Like previous steps, this allows calculation on each pixel of the phase map. The texture that coordinate calculation renders to needs

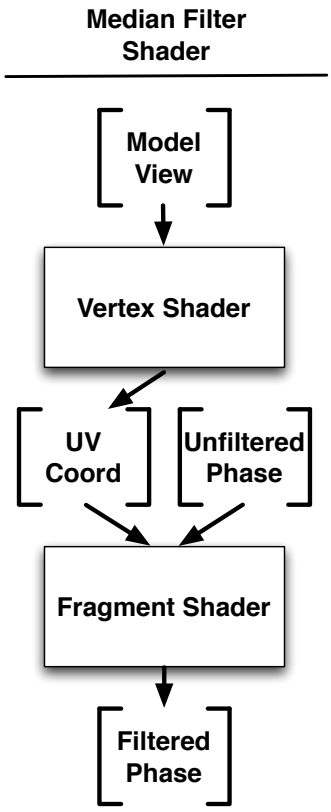


Figure 3.9 Holoimage median filter shader (McGuire, 2008). The Vertex Shader takes in the model view matrix and emits a UV coordinate. The Fragment Shader takes in the UV coordinate along with the unfiltered phase map, performs a 3×3 median filter, and then emits the filtered phase.

to be a floating point texture, and can be a single channel texture known as a luminance map. Once the FBO is bound, and the coordinate calculator shader is bound, the screen aligned quad is rendered, calling the Fragment Shader on each pixel in the phase map.

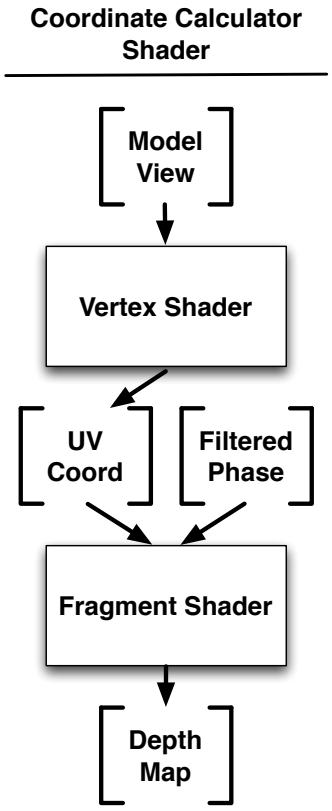


Figure 3.10 Holoimage coordinate calculator shader. The Vertex Shader takes in the model view matrix and emits a UV coordinate. The Fragment Shader takes the UV coordinate and filtered phase, applies Equations (3.20)-(3.22), and then emits the depth map.

Coordinate calculation works by applying Equations (3.20)-(3.22). This process is shown by Figure 3.10. Again like previous steps the Vertex Shader simply applies the model view projection matrix to the vertex, and varies the texture coordinate. The coordinate calculation is performed in the Fragment Shader. To calculate the x used in decoding, Equation (3.21) is used, along with the U component of the texture coordinates. Equation (3.21) simply yields the texture coordinate normalized to the range of

$(0, 1)$. Since texture coordinates are already normalized to this range s may be directly used for x, as well as V being directly used for y. Next the phase value is read in from the filtered phase map, and Equation (3.20) is applied. For the values of P, θ, and W these may be passed in as uniform variables allowing for them to flexibly change with different Holoimages. The resulting z is written out as the fragment color, yielding a floating point depth map, which can be used to render the geometry.

3.1.5.4 Normal Calculator

Without normals the lighting of the Holoimage will be missing. A normal map can be applied, allowing this stage of the pipeline to be skipped, or normals can be calculated. Assuming that normals are to be calculated, a normal map must be generated. This is once again done with the same FBO, screen aligned quad, and a different texture. The normal map texture used in the system developed for this research makes use of a floating point texture with three color channels. This allows the normalized normal vector to be saved to the texture, yielding a normal map for the final render stage. Once the FBO and normal calculator are bound, the screen aligned quad is rendered, calling the Fragment Shader for each pixel on the depth map.

Normal calculation works by averaging adjacent surface normals to generate a point normal for each point on the Holoimage. The shader process is shown with Figure 3.11. Again the Vertex Shader takes in the model view matrix and outputs a varying UV coordinate. Next the Fragment Shader takes in the UV coordinate along with the depth map performing normal calculation. To calculate a normal an array with vectors to each of a pixels neighbors is constructed, and then the cross product of successive indices is taken. The last vector is crossed with the beginning vector, which yields a surface normal for each of the neighboring polygons. All of these normals are averaged, and then the result is normalized, yielding a point normal for the index being evaluated. The result is written out as the fragment color, yielding a normalized normal map for each pixel in

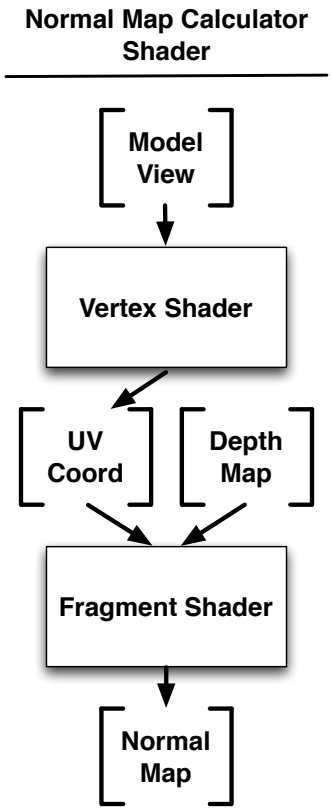


Figure 3.11 Holoimage normal calculator shader. The Vertex Shader takes in the model view matrix and emits a UV coordinate. The Fragment Shader takes in the UV coordinate along with the depth map, averages the surfaces normals for all adjacent polygons, and then emits a normal map.

the depth map.

3.1.5.5 Final Render

Finally, a depth map with accompanying normal map has been generated for each pixel in the Holoimage. At this step the FBO that was used for image processing is unbound, and the frame buffer is set to render to the previously used buffer. Also, the projection is switched back to a perspective projection, although an orthographic projection could be used. The final render shader is bound, and a plane of pixels uniformly spaced for each pixel in the Holoimage is rendered. This causes the Vertex Shader to be called for each pixel, along with the Fragment Shader to be called for each pixel.

Figure 3.12 shows the final render shader process. In the Vertex Shader, for each vertex. For a given vertex, the value in the coordinate map is read in, and the z component of the vertex is displaced by this amount. After this the model view projection transform is applied, and the texture coordinates are varied for the Fragment Shader. Inside the Fragment Shader, per pixel Phong shading is applied with the normal map that was previous generated. This yields the correctly shaded point for each pixel, which is written out as the fragment color.

3.2 Experimental Results

At this stage, 3D geometry has been successfully encoded with GLSL encoding shaders, saved to an image, read back in, and decoded with GLSL decoding shaders. To further verify this technique, the rest of this section performs the Holoimage technique on four test objects, a unit circle, a flat plane, a step height object, and multiple spheres. These objects are chosen as they are representative 3D objects, having features such as round and flat surfaces, sudden large height variations, and discontinuous surfaces.

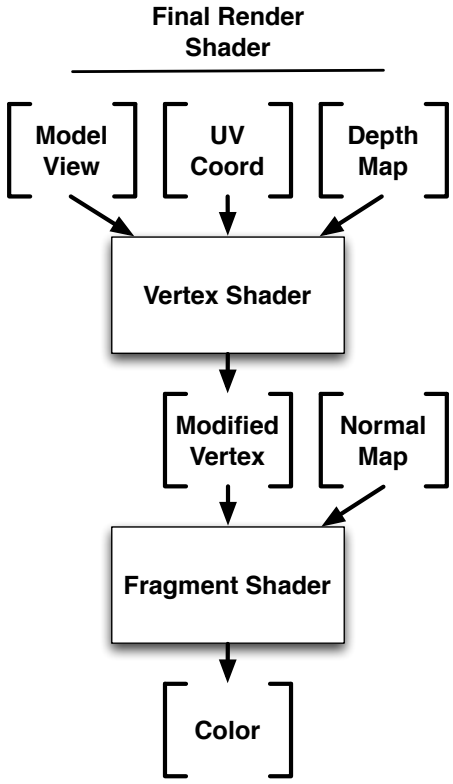


Figure 3.12 Holoimage final render shader. The Vertex Shader takes in the model view matrix, a UV coordinate, and the coordinate map, emitting a modified vertex according to the coordinate map. The Fragment Shader takes in the modified vertex along with the normal map and then applies per vertex Phong shading, emitting the colored fragment.

Figure 3.13 shows the four objects, with (a) - (d) showing the resulting Holoimage, (e) - (h) showing the reconstructed objects, and (i) - (l) showing a difference map of the ideal and reconstructed object. On the unit sphere there are some very small deviations which are a single pixel in size. This is due to quantization error, as the fringe is kept in a floating point format until it is saved to an image, at which point the floating point value ranging from $(0, 1)$ gets quantized into a byte in the range of $(0, 255)$. This is also shown on the plane, step height object, and multiple spheres. The deviation is subtle, but noticeable on flat surfaces or perfectly round surfaces when zoomed in.

Another type of error that shows up is subpixel shifting. Subpixel shifting occurs when an edge goes through the middle of a pixel in the Holoimage. This can be seen by looking at Figure 3.13 (k), which has a single horizontal line. This line occurs between the second and third step of the step height object. Since this edge goes though the middle of a set of pixels in its Holoimage there is an error with an incorrectly encoded height. In the recovered object, Figure 3.13 (g) it is not noticed since it just blends in with the step.

To quantify the amount of error being observed, root mean squared error (RMSE) was calculated using Equation (3.23). When this error is calculated, any pixel that is background in ideal figure and the Holoimage is masked off. This is done so that error calculation is only performed on the actual shape. As can be seen in Table 3.1, the error in each figure is less than 1%. Most off the error is actually less than 0.1%, except for the step height object which has error along the second and third step due to subpixel shifting. This shows that the Holoimage technique can successfully encoding and decode geometry with little error.

$$NRMSE = \sqrt{\frac{(\sum_{i=0}^{n}[Z_{Ideal} - Z_{Observed}]^2) - n\bar{Z}^2_{observed}}{n-1}}. \tag{3.23}$$

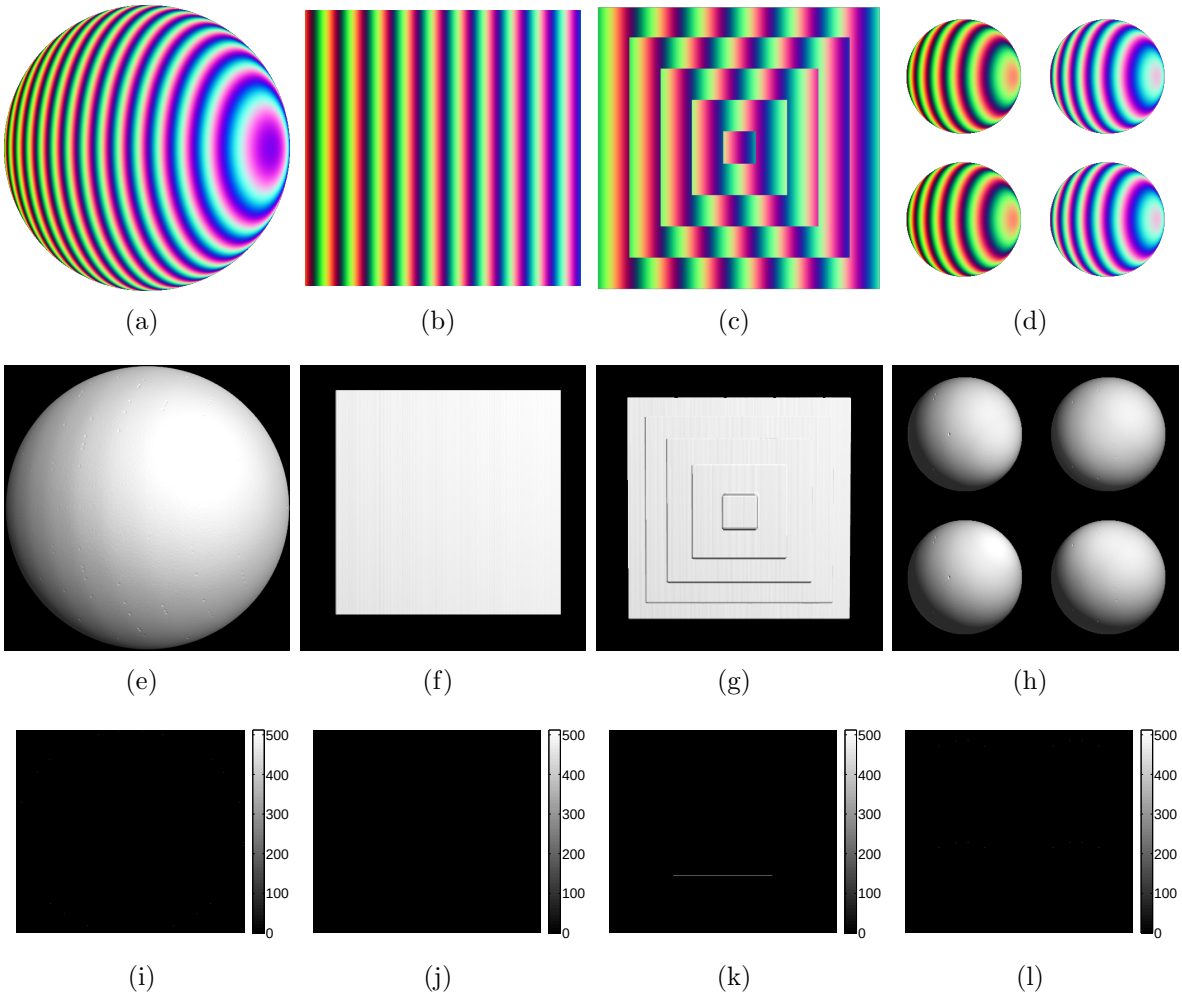


Figure 3.13 Sample renderings of objects (unit sphere, plane, step height block, and multiple spheres) that have been encoded and decoded with the Holoimage technique. The (a - d) contains the Holoimages that were generated, (e - h) contains the resulting renders, and (i - l) contains the difference map between the ideal and reconstructed figures. Note the resolution of the Holoimage used was 512×512, an $f = 6$ was used within a viewing volume of x varying from $(-1, 1)$ thus the fringe frequency was 12, and $\theta = 30°$.The viewing volume was changed to fit a unit sphere, which requires a simple scaling factor in Equations (3.21)-(3.22).

	Plane	Sphere	Multiple Spheres	Step Height
RMSE	0.01%	0.07%	0.01%	0.93%

Table 3.1 The error due to encoding the test objects. The largest error, the step height blocks error, is due to subpixel shifting along the bottom of the second and third step, shown in Figure 3.13 (k).

3.3 Summary

Holoimage has the ability to effectively encode and decode 3D geometry into a single 2D image. Since all of steps only require local pixel information, both the encoding and decoding process can be performed in parallel. This chapter demonstrated the principle behind both encoding and decoding, and then showed how encoding and decoding could be performed in parallel using the OpenGL Shading Language (GLSL). Finally sample renderings of objects that were Holoimage were shown, proving that the technique not only works in theory, but in actuality.

CHAPTER 4. HOLOIMAGE COMPRESSION

In the ideal lossless case, the Holoimage technique stores 3D geometry as a 2D image with little loss of quality, less than 0.1% root mean squared error (RMSE). Although this is good, many 2D image formats make use of lossy encoding to further encode visual information with small artifacts that are typically not noticed. This chapter explores how to store a Holoimage in lossy Joint Photograph Experts Group (JPEG) format. Three different experiments are performed on the pattern used by Holoimage to create a pattern that is resilient to artifacts introduced by JPEG compression.

4.1 Experiments

Now that Holoimages can be generated, and ideal lossless ones can be saved with little error (approximately 0.01% RMSE), this chapter looks at how lossy compression affects the Holoimage. The first experiment looks at how JPEG compression affects the red and green color channels, and how changes in the frequency of the fringe affect the RMSE. The second experiment looks at how to encode stair step K from Equation (3.4) in the blue color channel, and does this by creating large distances between successive values of K through a step height algorithm. Finally the third experiment takes findings from experiment two, and embeds a cosine wave into the step height algorithm, making it more resilient to JPEG compression. In all of these experiments a Holoimage of 512×512 pixels is used that is rotated 30° about the y axis.

4.2 Varying Frequency of Fringe Pattern

The first experiment is to evaluate how variations in the frequency of the encoded fringe pattern affect the reconstruction of the geometry, in a lossless format and a lossy format. The fringe in the red and green color channel is given with Equations (3.1)-(3.2) where f is the frequency of the fringe, and x is the x position of the geometry. Figure 4.1 shows a plot of the fringe in the red and green color channels. In this experiment the encoding of K is in the blue color channel and is kept in a lossless format to ensure that only variations in the fringe pattern are having an effect. Frequencies ranging from 1 to 16 were tested on a unit sphere, a plane, a stair height block, and four separate spheres.

Masks were created by checking every pixel for a color value and assigning a value of either 1 or 0. If a color value existed (i.e. not 0) in either the Holoimage in review or in the ideal image, then it was considered part of the geometry and given value 1; all other pixels were assigned 0. After this, erosion, an image morphology operation, was applied with a disk shaped structuring element that was one unit wide. This removed a single pixel around the entire border of the shape, to reduce the noise on the boundary. From here the RMSE was calculated and results were analyzed.

The hypothesis is that there is an optimal frequency, which will minimize error in the reconstructed object when compared to the original. This will be found by Holoimaging the test shapes and calculating the root mean squared error. Frequencies that are less than and greater than this optimal frequency should result in a larger RMSE.

4.2.1 Results

The hypothesis for the varying frequency of the fringe pattern was that there would be an optimal frequency for the fringe in the red and green color channels. Tables 4.1-4.4 give the results for the unit sphere object, flat plane, step height block, and multiple spheres, respectively. Figure 4.3 shows a sample rendering of the unit sphere after

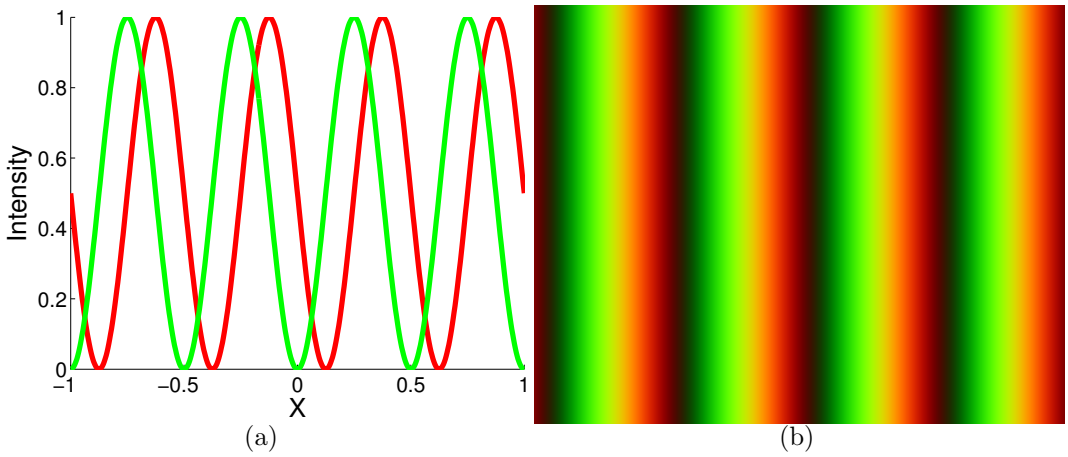


Figure 4.1 The encoded structured pattern used to test the varying fringe frequency. (a) single cross section of the pattern, with the x position of the geometry along the x axis and color intensity in the red and green color channels along the y axis. (b) the structured pattern plotted as a flat texture. Note that a fringe frequency $f = 2$ was used.

encoding. The following sections will look in detail at the results of each figure, discussing sources of error.

4.2.1.1 Unit Sphere

Quantization error is more prevalent in the lower frequencies such as $f = 1$ and $f = 2$. This makes sense, as small errors in the fringe such as those associated with quantization, have a much higher impact in the unwrapped phase maps. Figure 4.2 shows an example of this with the quantized unit sphere with an $f = 1$. There is a spot on the right of the sphere which is erroneously encoded. Rounding errors due to quantization in the red and green channels actually make the correct $K = 2$ but as in the ideal case it should actually be $K = 1$. This is because this spot is near the far right of the projectors projecting volume, right next to the phase shift. As f increases, this quantization error is reduced since larger jumps in the fringe are needed to create such

an error. Looking at the Figure 4.2, this spot gets smaller and smaller until $f = 6$ is achieved. At this frequency and higher quantization error has little impact.

The next area of error occurs from under sampling the encoding pattern. As the frequency gets higher and higher, recreating the sinusoidal fringe gets difficult, as the number of points used for reconstructing a period of fringe goes down. Errors in the quantized Holoimage do not show this problem as much, but errors in the lossy encoded Holoimage show this quite a bit more. As $f >= 12$ this error is shown with small ripples on the surface of the reconstructed object. This is somewhat hard to see on the sphere but becomes more apparent on the plane.

Finally there is error due to lossy encoding of the Holoimage. Figure 4.3 shows the reconstruction of the unit sphere with $f = 2$ at varying levels of JPEG encoding. As the quality of the JPEG image goes down so does the Holoimage which is expected, but even at a quality level of 75, once f is greater than 2 there is less than 2% error. Again like the error due to quantization, most of the error occurs next to the phase jumps, as the fringe goes gets rounded above or below the correct value of K.

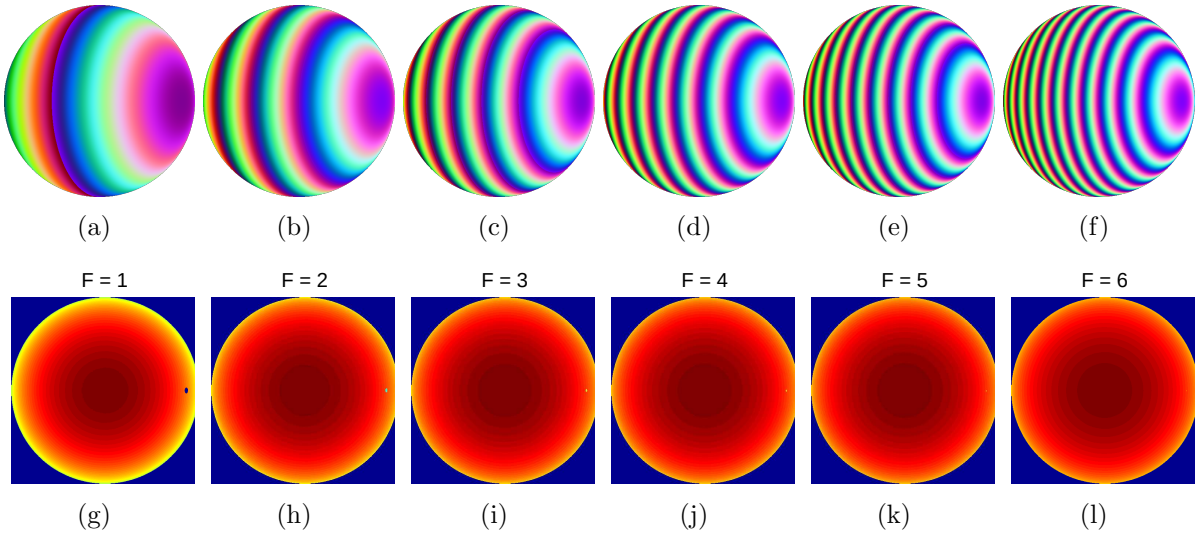
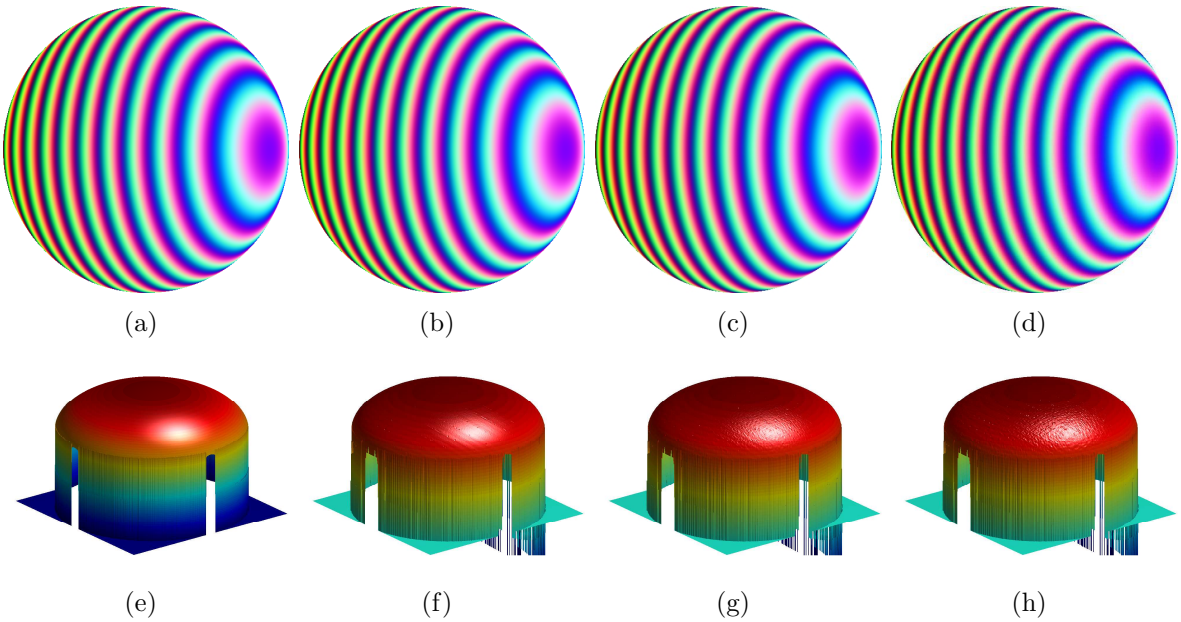


Figure 4.2 Reconstructed unit sphere with varying frequency f. Note the small error spot on the right of the sphere which shrinks as f increases.



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

(e) (f) (g) (h)

Figure 4.3 Samples of the reconstructed sphere. (a) ideal sphere figure (b) Holoimage encoded and kept in lossless png format (c) Holoimage encoded in JPEG level 100, (d) Holoimage encoded in JPEG level 75. Note that a fringe frequency $f = 6$ was used.

Unit Sphere	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	2.35%	2.99%	3.92%	5.33%	6.96%	7.42%	6.98%
$f = 2$	0.76%	1.24%	1.24%	1.28%	1.46%	1.44%	1.64%
$f = 3$	0.36%	1.36%	1.34%	1.26%	1.30%	1.24%	1.37%
$f = 4$	0.20%	1.03%	1.01%	1.11%	1.02%	1.07%	1.15%
$f = 5$	0.13%	0.93%	0.93%	0.93%	0.94%	0.94%	0.96%
$f = 6$	0.07%	1.03%	1.03%	1.04%	1.05%	1.04%	1.04%
$f = 7$	0.05%	1.00%	1.00%	1.00%	1.00%	1.01%	1.00%
$f = 8$	0.02%	0.78%	0.78%	0.79%	0.79%	0.78%	0.79%
$f = 9$	0.01%	0.98%	0.98%	0.99%	0.99%	0.99%	0.99%
$f = 10$	0.01%	1.00%	1.00%	1.00%	0.99%	0.99%	0.99%
$f = 11$	0.01%	0.84%	0.84%	0.84%	0.84%	0.85%	0.85%
$f = 12$	0.03%	0.92%	0.92%	0.92%	0.93%	0.93%	0.93%
$f = 13$	0.01%	0.76%	0.76%	0.76%	0.76%	0.76%	0.76%
$f = 14$	0.01%	0.81%	0.80%	0.81%	0.81%	0.81%	0.81%
$f = 15$	0.01%	0.91%	0.91%	0.91%	0.91%	0.91%	0.91%
$f = 16$	0.01%	0.87%	0.87%	0.87%	0.87%	0.87%	0.87%

Table 4.1 RMSE of encoding a unit circle. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

4.2.1.2 Plane, Step Height Block, and Multiple Spheres

As seen previously with the unit sphere there is more error in the lower frequencies, but it is minimized after $f > 4$. Figure 4.4 shows an example of the quantized objects at $f = 10$. Tables 4.2-4.4, give the RMSE for the plane, step height block, and multiple spheres, respectively. One thing to note is that in the lower frequencies there are some very small ripples on the plane and step height block, which occurs from quantization. Small minute differences change the objects flat surfaces from being perfectly flat, to having small ripples which are accentuated by lighting. Since the point normals are calculated by averaging all adjacent surface normals, this error persists into the normal map. If a normal map created from the ideal figure was used, it would minimize this effect.

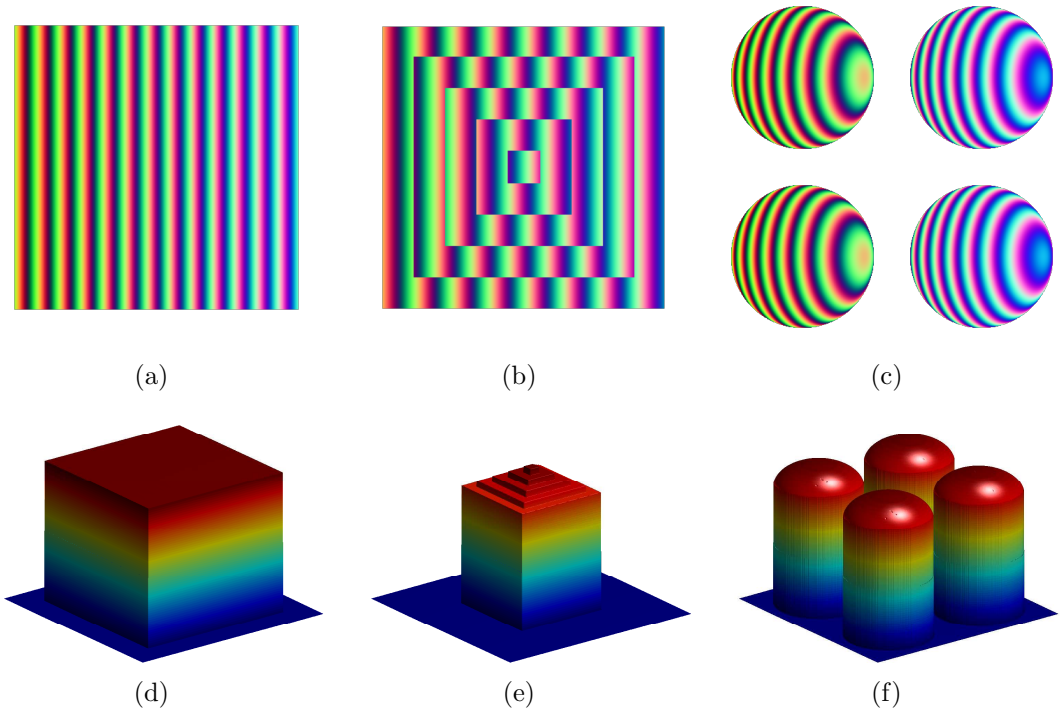


Figure 4.4 Plane, Step Height Block, and Multiple Spheres quantized with $f = 10$. The (a)-(c) is the quantized holoimage, and (d)-(f) bottom row contains the reconstructed object.

Examining Table 4.3, it can be seen that the percentage of error with the step height

block is higher than it is with the other objects. This error comes from the step along the second and third step of the block and is caused by subpixel shift, as explained in Section 3.2. Since the depth is encoded in the pixel values of the Holoimage, the resolution is limited by the resolution of the Holoimage. If the edge does not perfectly line up with the edge of a pixel, there is a slight shift in the edge, and this shift is called subpixel shift. By using a higher resolution Holoimage, this error could be mitigated.

Finally, error due to the lossy encoding is reduced when $f > 4$. Similar to the quantization, the error is reduced with higher frequency fringe since the small variations in the fringe have less impact on the resulting wrapped phase map. Once the frequency gets too high, error due to under sampling the encoding pattern occurs which is shown by the lossy encoding. Although very minimal, slight rises in error occur when $f = 16$.

Plane	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.07%	1.13%	1.13%	1.14%	1.14%	1.14%	1.14%
$f = 2$	0.04%	0.60%	0.60%	0.60%	0.67%	0.62%	0.66%
$f = 3$	0.03%	0.40%	0.40%	0.40%	0.40%	0.40%	0.40%
$f = 4$	0.02%	0.28%	0.28%	0.28%	0.28%	0.28%	0.28%
$f = 5$	0.01%	0.22%	0.24%	0.22%	0.26%	0.26%	0.26%
$f = 6$	0.01%	0.16%	0.20%	0.14%	0.16%	0.16%	0.16%
$f = 7$	0.01%	0.14%	0.14%	0.13%	0.13%	0.18%	0.17%
$f = 8$	0.01%	0.02%	0.03%	0.03%	0.03%	0.09%	0.05%
$f = 9$	0.01%	0.09%	0.10%	0.09%	0.08%	0.11%	0.04%
$f = 10$	0.01%	0.07%	0.03%	0.03%	0.07%	0.10%	0.08%
$f = 11$	0.02%	0.06%	0.09%	0.06%	0.09%	0.07%	0.09%
$f = 12$	0.01%	0.08%	0.08%	0.08%	0.08%	0.10%	0.09%
$f = 13$	0.01%	0.08%	0.08%	0.08%	0.09%	0.08%	0.06%
$f = 14$	0.01%	0.08%	0.08%	0.08%	0.10%	0.07%	0.06%
$f = 15$	0.00%	0.07%	0.07%	0.08%	0.05%	0.07%	0.08%
$f = 16$	0.00%	0.06%	0.06%	0.06%	0.06%	0.07%	0.08%

Table 4.2 RMSE of encoding a plane. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

4.2.2 Experiment Summary

Varying the frequency of the fringe pattern in the red and green channel caused error in the lower frequencies, and the error is reduced in the higher frequencies. The data

Step Height	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.71%	9.05%	7.47%	11.91%	12.59%	10.50%	11.81%
$f = 2$	0.49%	6.50%	6.51%	6.51%	6.55%	6.55%	6.50%
$f = 3$	0.04%	6.47%	6.47%	6.47%	6.48%	6.51%	6.48%
$f = 4$	0.13%	6.62%	6.62%	6.64%	6.63%	6.64%	6.66%
$f = 5$	0.04%	6.71%	6.71%	6.72%	6.72%	6.71%	6.71%
$f = 6$	0.05%	6.71%	6.71%	6.70%	6.70%	6.71%	6.71%
$f = 7$	0.09%	6.79%	6.79%	6.79%	6.79%	6.79%	6.79%
$f = 8$	0.11%	6.70%	6.70%	6.70%	6.70%	6.70%	6.67%
$f = 9$	0.05%	6.75%	6.75%	6.75%	6.75%	6.75%	6.75%
$f = 10$	0.04%	6.76%	6.76%	6.77%	6.77%	6.76%	6.74%
$f = 11$	0.04%	6.76%	6.76%	6.76%	6.75%	6.75%	6.74%
$f = 12$	0.02%	6.78%	6.79%	6.79%	6.78%	6.78%	6.78%
$f = 13$	0.04%	6.77%	6.77%	6.77%	6.77%	6.77%	6.77%
$f = 14$	0.04%	6.82%	6.82%	6.81%	6.82%	6.81%	6.82%
$f = 15$	0.02%	6.80%	6.80%	6.80%	6.80%	6.80%	6.79%
$f = 16$	0.06%	6.82%	6.82%	6.81%	6.82%	6.82%	6.82%

Table 4.3 RMSE of encoding a step height block. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

Multiple Circles	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.07%	1.44%	1.45%	1.58%	1.69%	1.64%	1.75%
$f = 2$	0.04%	1.84%	1.83%	1.87%	1.93%	1.89%	1.90%
$f = 3$	0.03%	1.87%	1.88%	1.91%	1.90%	1.91%	1.95%
$f = 4$	0.02%	1.78%	1.79%	1.74%	1.79%	1.77%	1.78%
$f = 5$	0.02%	1.66%	1.65%	1.67%	1.69%	1.68%	1.71%
$f = 6$	0.01%	1.74%	1.75%	1.75%	1.74%	1.73%	1.75%
$f = 7$	0.01%	1.73%	1.73%	1.73%	1.73%	1.74%	1.74%
$f = 8$	0.01%	1.72%	1.72%	1.73%	1.73%	1.73%	1.72%
$f = 9$	0.01%	1.73%	1.73%	1.73%	1.73%	1.73%	1.74%
$f = 10$	0.01%	1.50%	1.50%	1.50%	1.50%	1.49%	1.50%
$f = 11$	0.01%	1.69%	1.70%	1.69%	1.70%	1.69%	1.70%
$f = 12$	0.01%	1.70%	1.70%	1.70%	1.70%	1.70%	1.70%
$f = 13$	0.01%	1.58%	1.58%	1.58%	1.58%	1.58%	1.58%
$f = 14$	0.01%	1.59%	1.59%	1.59%	1.59%	1.59%	1.59%
$f = 15$	0.01%	1.53%	1.53%	1.53%	1.53%	1.53%	1.53%
$f = 16$	0.01%	1.67%	1.67%	1.67%	1.67%	1.67%	1.68%

Table 4.4 RMSE of encoding multiple spheres. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

showed that once the frequency f was above 6, quantization error had little effect and lossy encoding error could be kept around 1%. From the data, the optimal frequency seems to be between $8 <= f <= 12$. This is the range that has the smallest error across all the objects. The object that had the largest error was the step height block which was due to subpixel shifting along one of the steps. JPEG encoding increases this error since it performs the best on what is know as natural images, thus sharp discontinuities do not encode very well. If another image encoding technique that preserved the discontinuities was used, this error could be further reduced.

4.3 Varying Encoding of K for Phase Unwrapping

In the previous experiment the encoding of K in the blue color channel was kept in a lossless format so that error analysis on the fringe could be performed without compounding it with the error due to encoding K. The next two experiments look at different encodings of K so that it can be put into a lossy format along with the fringe patterns. For each of these experiments the red and green color channels are kept in a lossless floating point format so that there is no quantization error.

4.3.1 Step Height Encoding of K

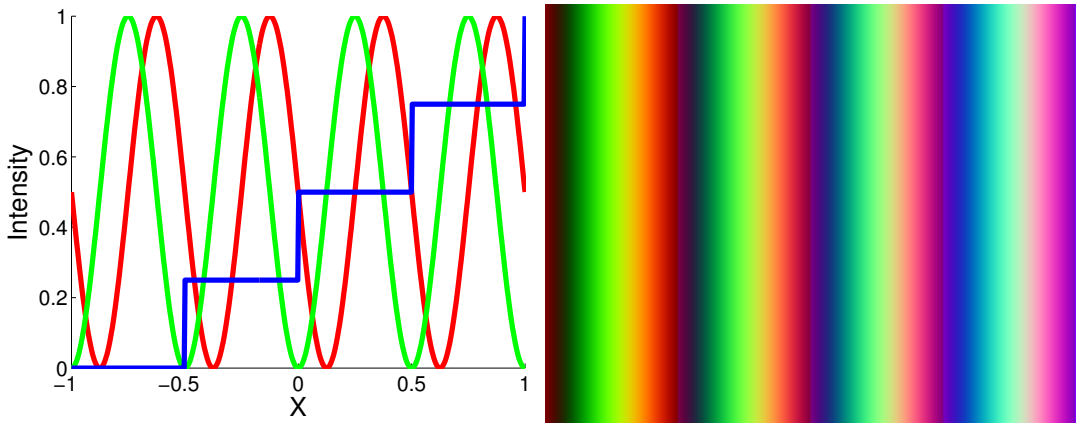
In this experiment, the value of K used for phase unwrapping is multiplied by a step height and then encoded in the blue color channel. The step height is chosen such that it can be the largest possible with uniform height variations for each value of K. Figure 4.5 gives an example of the step height encoding of K in the blue channel along with the red and green channels. The encoding of K is given by

$$I_b = floor(x \times f) \times stepHeight + 0.5, \tag{4.1}$$

$$K = floor(\frac{I_b}{stepHeight}), \tag{4.2}$$

$$stepHeight = \frac{1.0}{2.0 \times f}. \tag{4.3}$$

Where x is the x position of the geometry, f is the frequency of the fringe, stepHeight is the height increment between successive values of K, and I_b is the value to be used in the blue channel.



(a) (b)

Figure 4.5 The encoded structured pattern used to test the step height encoding of K. (a) shows a single row of the pattern, with the x position of the geometry along the x axis and color intensity in the red green and blue color channels along the y axis. (b) shows the structured pattern plotted as a flat texture. Note that a fringe frequency $f = 2$ was used, with a $stepHeight = \frac{1}{4}$.

The hypothesis for this experiment is that lossy compression will not significantly affect the encoding of K in the middle of a step, but will have an effect near the K jumps. This will be tested by varying levels of JPEG compression on the the unit sphere, plane, step height block, and four separate spheres. Again RMSE will be calculated, compared to the actual geometry, along with visual comparisons of the reconstructed geometry.

4.3.2 Results

The hypothesis for this experiment was that lossy encoding would not have a large effect in the middle of the step but would have a large effect along the steps, or phase changes of the geometry. Tables 4.5-4.8 give the results for the unit sphere object, flat plane, step height block, and multiple spheres, respectively. Figure 4.6 shows a sample rendering of the unit sphere after encoding. The following sections will look in detail at the results of each figure, discussing sources of error.

4.3.2.1 Unit Sphere

Quantization error had a very small effect on the reconstruction of the object when K was encoded into the blue color channel using the step height encoding. While performing the experiments using Equation (4.3) to generate K, it was found that many of the results were off by a single phase jump due to quantization and would have large errors, greater than 80% due to this fact. To circumvent this problem, a modified equation was substituted.

$$K^* = floor(\frac{I_b + \frac{1}{2^8}}{stepHeight}). \tag{4.4}$$

This equation adds the amount that the blue channel could be off due to quantization. This removed most of the quantization error and was then used in the rest of the experiments.

As f increased, error slightly rose in the quantized figures. This is due to the fact that the step width and the step height shrunk. Therefore, a slight deviation had more impact, because the step height was smaller. This seemed to have little effect, as the error in the quantized object never went over 0.03%.

Error due to lossy encoding had a much more profound effect. The JPEG encoded columns of Table 4.5 show this, along with Figure 4.6. Unlike the lossy encoding of the

red and green channels, ripples are no longer an effect. Instead there is spiking error, which comes from incorrect values of K during phase unwrapping. Since K is off by at least an entire phase jump, an error of at least 2π is introduced into the phase map. This leads to large depth distance discrepancies during reconstruction which leads to spiking noise. As f increases this error seems to go down, and then goes up again as it gets to high. Approximately at $f = 10$ seems to be where the error is minimized in terms of lossy encoding. The surface of the reconstructed object is no longer acceptable though as is shown by Figure 4.6.

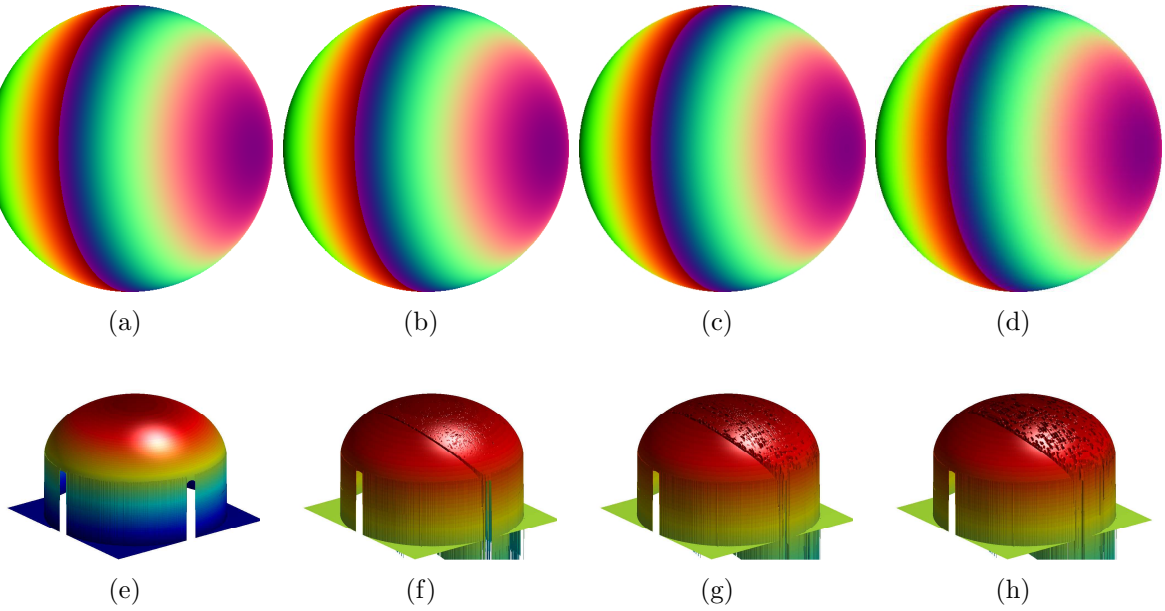


Figure 4.6 Unit circle under varying levels of JPEG compression with step height encoding of K. The left figure is the ideal, and then moving right is JPEG quality level 100, 90, and 75. Note all of the holes that are generated in the reconstructed object due to the lossy encoding of K in the blue channel. Fringe frequency $f = 1$ was used with a $stepHeight = \frac{1}{2}$.

4.3.2.2 Plane, Step Height Block, and Multiple Spheres

As has been seen with the unit sphere, errors due to quantization have a very small effect on K and the reconstruction. The only object that it seems to have a large effect

Unit Sphere	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.00%	9.33%	15.04%	19.48%	23.10%	26.22%	29.30%
$f = 2$	0.00%	7.13%	11.38%	14.91%	17.50%	19.06%	20.61%
$f = 3$	0.00%	10.07%	12.47%	13.89%	14.93%	15.44%	15.77%
$f = 4$	0.00%	5.88%	8.01%	9.30%	10.43%	11.10%	11.33%
$f = 5$	0.01%	7.00%	8.27%	9.00%	9.38%	9.71%	9.90%
$f = 6$	0.01%	6.96%	7.68%	8.07%	8.31%	8.50%	8.65%
$f = 7$	0.01%	6.80%	7.14%	7.32%	7.56%	7.71%	7.81%
$f = 8$	0.03%	5.66%	6.00%	6.31%	6.61%	6.85%	7.01%
$f = 9$	0.02%	6.13%	6.27%	6.39%	6.54%	6.74%	6.91%
$f = 10$	0.01%	5.97%	6.04%	6.18%	6.35%	6.57%	6.77%
$f = 11$	0.01%	6.09%	6.15%	6.24%	6.41%	6.68%	6.86%
$f = 12$	0.01%	6.21%	6.25%	6.38%	6.56%	6.80%	7.05%
$f = 13$	0.01%	6.42%	6.46%	6.60%	6.78%	7.02%	7.39%
$f = 14$	0.01%	6.67%	6.70%	6.85%	7.07%	7.37%	7.70%
$f = 15$	0.01%	7.08%	7.13%	7.27%	7.48%	7.79%	8.16%
$f = 16$	0.02%	7.43%	7.47%	7.66%	7.90%	8.15%	8.52%

Table 4.5 RMSE of encoding the unit circle with the step height encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

on is the step height block. Subpixel shifting along the step variations is where this error is coming from.

As with the unit circle, error due to lossy encoding presents. Looking at the data in Tables 4.6-4.8 $f = 10$ is where the error is minimized. As was seen with the unit circle the error going down as f increases is due to 2π errors in the phase map not having as profound effect. Error going up as $f > 10$ is resulting from the step height and step width being so small that the errors are becoming more frequent.

4.3.3 Experiment Summary

Encoding K in the blue color channel works when the image medium is lossless. Quantization error was shown to be under 1%, but this technique fails when the image medium is lossy. Error for JPEG encoding at quality level 100 was over 6% in most cases, and resulted in reconstructed objects with large amounts of spiking error. Even median filtering was unable to remove all of the spiking noise, thus a different method

Plane	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.00%	7.95%	9.42%	6.27%	6.28%	19.04%	18.38%
$f = 2$	0.00%	3.43%	6.58%	14.09%	16.55%	18.51%	19.78%
$f = 3$	0.00%	6.76%	10.08%	10.69%	12.45%	15.19%	14.30%
$f = 4$	0.00%	1.29%	5.88%	6.66%	9.66%	10.70%	9.98%
$f = 5$	0.00%	6.29%	7.24%	8.38%	8.58%	9.21%	9.48%
$f = 6$	0.00%	5.58%	6.47%	7.47%	7.62%	7.72%	7.91%
$f = 7$	0.00%	4.94%	5.46%	6.07%	6.59%	6.76%	6.67%
$f = 8$	0.00%	4.35%	4.85%	5.23%	5.81%	5.92%	6.02%
$f = 9$	0.00%	4.90%	4.75%	4.87%	5.05%	5.15%	5.29%
$f = 10$	0.00%	4.33%	4.48%	4.52%	4.60%	4.76%	4.71%
$f = 11$	0.00%	4.77%	4.80%	4.60%	4.63%	4.79%	4.89%
$f = 12$	0.00%	3.93%	3.99%	3.82%	3.94%	4.07%	4.09%
$f = 13$	0.00%	3.71%	3.77%	3.79%	3.65%	3.82%	4.01%
$f = 14$	0.00%	4.16%	4.19%	4.18%	4.05%	4.53%	4.73%
$f = 15$	0.00%	3.31%	3.28%	3.26%	3.69%	4.04%	4.15%
$f = 16$	0.00%	3.15%	3.14%	3.18%	3.86%	4.17%	4.42%

Table 4.6 RMSE of encoding a plane with the step height encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

Step Height	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.63%	15.65%	23.74%	29.51%	29.92%	32.47%	37.16%
$f = 2$	0.70%	9.93%	12.93%	17.84%	19.37%	20.94%	21.12%
$f = 3$	0.93%	10.73%	13.28%	14.43%	15.22%	15.88%	16.31%
$f = 4$	0.85%	6.49%	9.19%	9.91%	11.14%	11.54%	11.95%
$f = 5$	0.96%	6.40%	8.25%	9.40%	9.92%	10.03%	10.56%
$f = 6$	0.93%	7.46%	8.11%	8.80%	8.69%	9.13%	9.19%
$f = 7$	0.89%	7.89%	8.35%	8.71%	8.85%	8.65%	8.83%
$f = 8$	0.95%	6.63%	7.17%	7.27%	7.42%	7.82%	8.11%
$f = 9$	0.94%	6.73%	6.82%	6.98%	7.19%	7.54%	7.79%
$f = 10$	0.96%	6.84%	6.99%	6.99%	7.15%	7.22%	7.57%
$f = 11$	0.95%	6.46%	6.55%	6.62%	6.70%	6.92%	7.34%
$f = 12$	0.98%	6.82%	6.79%	6.91%	6.96%	7.33%	7.84%
$f = 13$	0.95%	6.78%	6.92%	6.84%	6.67%	7.08%	7.61%
$f = 14$	0.94%	6.39%	6.37%	6.54%	6.80%	7.22%	7.57%
$f = 15$	0.97%	5.99%	6.05%	6.11%	6.42%	6.80%	7.18%
$f = 16$	0.95%	6.08%	6.10%	6.34%	6.50%	6.74%	7.09%

Table 4.7 RMSE of encoding the step height block with the step height encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

Multiple Circles	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 1$	0.00%	8.19%	13.39%	16.93%	21.57%	23.79%	25.08%
$f = 2$	0.00%	7.53%	11.75%	14.72%	17.52%	19.52%	20.13%
$f = 3$	0.00%	10.39%	13.07%	14.31%	15.08%	15.66%	15.93%
$f = 4$	0.00%	6.67%	8.64%	10.15%	10.77%	11.47%	11.99%
$f = 5$	0.00%	7.18%	8.30%	8.97%	9.48%	9.80%	10.01%
$f = 6$	0.00%	6.92%	7.64%	7.97%	8.31%	8.50%	8.73%
$f = 7$	0.00%	6.69%	7.10%	7.31%	7.56%	7.74%	7.98%
$f = 8$	0.00%	6.14%	6.45%	6.78%	7.18%	7.42%	7.67%
$f = 9$	0.00%	6.27%	6.44%	6.66%	6.93%	7.27%	7.51%
$f = 10$	0.01%	6.17%	6.26%	6.51%	6.72%	7.02%	7.24%
$f = 11$	0.00%	6.09%	6.18%	6.34%	6.57%	6.77%	7.22%
$f = 12$	0.00%	6.18%	6.22%	6.53%	6.67%	7.04%	7.31%
$f = 13$	0.00%	6.56%	6.66%	6.88%	7.15%	7.49%	7.81%
$f = 14$	0.01%	6.56%	6.68%	6.99%	7.23%	7.62%	7.87%
$f = 15$	0.01%	7.09%	7.16%	7.41%	7.60%	8.03%	8.53%
$f = 16$	0.00%	7.51%	7.59%	7.75%	8.08%	8.38%	8.72%

Table 4.8 RMSE of encoding multiple spheres with the step height encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

needed to be investigated.

4.3.4 Wavy Step Height Encoding of K

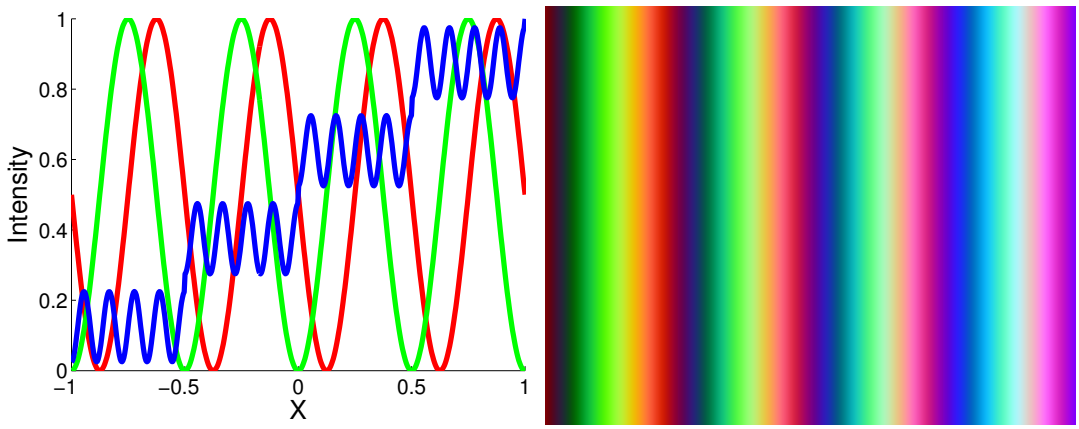
In this experiment a cosine function is embedded in the step height encoding so that each step oscillates versus having a constant value. The encoding of K is given by Equations (4.5)-(4.7) and an example of the fringe color channels is shown in figure 4.7.

$$I_b = \cos(\omega_{stair} \times (x - (\frac{K \times stepWidth}{S})) + \pi) \times (\frac{S}{2.5}) + (\frac{S}{2.0}) + K, \quad (4.5)$$

$$\omega_{stair} = f \times (f_{stair} + 0.5), \quad (4.6)$$

$$S = \frac{1.0}{2.0 \times f}. \quad (4.7)$$

The hypothesis is that since the value of K is encoded using cosine waves, it will encode better with the discrete cosine transform that is applied in JPEG compression, resulting in less error on the reconstructed figures. Since K is encoded with cosine



(a) (b)

Figure 4.7 The encoded structured pattern used to test the wavy step encoding of K. (a) shows a single row of the pattern, with the x position of the geometry along the x axis and color intensity in the red green and blue color channels along the y axis. (b) shows the structured pattern plotted as a flat texture. Note that a fringe frequency f of 2 was used and a stair fringe frequency f_{stair} of 4 was used.

functions in each step, the frequency of the waves can be adjusted. The second part of the hypothesis is that there is an optimal frequency of fringe; frequencies below and above this optimal range should result in higher root mean squared error. To test this hypothesis, frequencies ranging from $6 <= f <= 8$ for the fringe were tested and frequencies for the stairs $1 <= f_{stair} <= 4$ from were tested, again using a unit sphere, a stair height block, and four separate spheres.

4.3.5 Results

The hypothesis for this experiment was that by encoding K within a cosine wave, lossy encoding would not have as large of an effect on the overall error and the spiking error along the phase changes. Tables 4.9-4.12 give the results for the unit sphere object, flat plane, step height block, and multiple spheres, respectively. Figure 4.8 shows a sample rendering of the unit sphere after encoding. The following sections will look in

detail at the results of each figure, discussing sources of error.

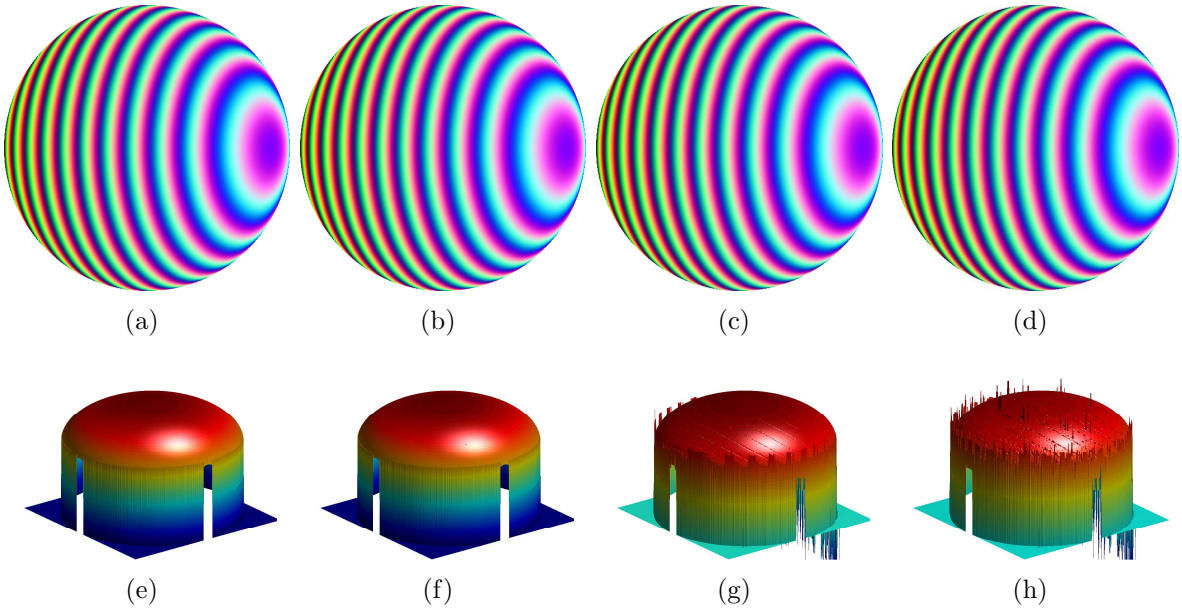


Figure 4.8 Unit circle under varying levels of JPEG compression with wavy encoding of K. (a) is the ideal, (b) is the quantized figure, (c) is JPEG quality level 100, and (d) is JPEG quality level 75. Note that most of the holes seen in the step height encoding of K are gone and most of the noise occurs around the boundary of the object. Note that a fringe frequency of $f = 6$, a stair fringe frequency $f_{stair} = 4$, and a $stepHeight = \frac{1}{12}$ was used.

4.3.5.1 Unit Sphere

As was seen with the step height encoding of K, quantization error had a very small effect on the reconstruction of the object. Comparing Table 4.5 to Table 4.9 the errors in the quantization are exactly the same. This is to be expected as quantization effects would only result if the stair height was too small, which for frequencies in the range of $[6, 8]$ this is not a problem. Error due to under sampling the encoding pattern was not as apparent as it was in previous experiments. Since f_{stair} is a multiple of f its frequency is multiplicative. For $f = 8$ and $f_{stair} = 4$ the frequency in the blue channel is 32. Since the Holoimage is 512 pixels wide, there is only 16 pixels which can make up an entire

period of the blue fringe. This would theoretically result in under sampling error, but it does not seem to be very apparent.

Error due to lossy encoding has gone down by a factor of 2 in some frequencies. Again comparing Table 4.5 to Table 4.9 the error is noticeably different. JPEG quality level 100 for $f = 6$ was 6.96% and has gone down to 3.13% with $f_{stair} = 4$. Figure 4.9 gives a comparison of the step height encoding of K at $f = 6$ compared to the wavy encoding of K. Furthermore, looking at the reconstructed figure it seems that most of the error is around the edge of the sphere, and that the phase jumps are close to being intact. With more median filtering performed on the phase map, what remains of the spiking error could be removed.

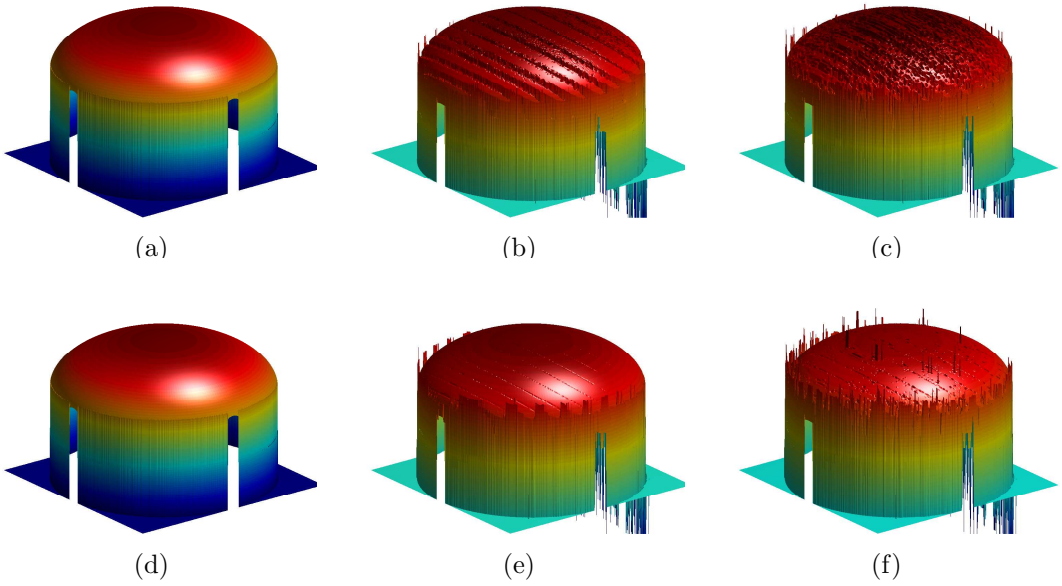


Figure 4.9 Comparison of the unit circle at $f = 6$. The(a)-(c) gives the step height encoding of K reconstruction, and (d)-(f) gives the wavy encoding of K reconstruction. (a) and (d) are the ideal, (b) and (e) are JPEG quality level 100, and (c) and (f) are JPEG quality level 75. Note that a $stepHeight = \frac{1}{12}$ was used.

Unit Sphere	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 6,\ f_{stair} = 1$	0.01%	3.24%	3.51%	3.89%	4.34%	4.74%	5.10%
$f = 6,\ f_{stair} = 2$	0.01%	3.20%	3.33%	3.61%	4.05%	4.46%	4.95%
$f = 6,\ f_{stair} = 3$	0.01%	3.13%	3.19%	3.38%	3.79%	4.12%	4.44%
$f = 6,\ f_{stair} = 4$	0.01%	3.13%	3.17%	3.38%	3.68%	4.00%	4.24%
$f = 7,\ f_{stair} = 1$	0.01%	3.67%	3.83%	4.23%	4.79%	5.29%	5.67%
$f = 7,\ f_{stair} = 2$	0.01%	3.56%	3.71%	4.13%	4.60%	4.93%	5.27%
$f = 7,\ f_{stair} = 3$	0.01%	3.59%	3.74%	4.00%	4.42%	4.68%	5.00%
$f = 7,\ f_{stair} = 4$	0.01%	3.61%	3.68%	3.95%	4.35%	4.64%	4.87%
$f = 8,\ f_{stair} = 1$	0.03%	4.02%	4.21%	4.49%	4.88%	5.26%	5.52%
$f = 8,\ f_{stair} = 2$	0.03%	3.81%	3.99%	4.28%	4.70%	5.13%	5.46%
$f = 8,\ f_{stair} = 3$	0.03%	3.73%	3.78%	4.11%	4.55%	4.95%	5.24%
$f = 8,\ f_{stair} = 4$	0.03%	3.72%	3.82%	4.09%	4.45%	4.76%	5.06%

Table 4.9 RMSE of encoding the unit circle with the wavy encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

4.3.5.2 Plane, Step Height Block, and Multiple Spheres

As was seen with the unit sphere, errors due to quantization have the same effect as they did with the step height encoding of K. Under sampling error is also not very apparent, and lossy error has been cut in half in most of the frequencies. The only object which did not drastically improve in error was the step height block which only seemed to improve slightly. This is still due to the subpixel shifting that is occurring on the boundary of the second and third step height variation. Comparing the results in Tables 4.10-4.12 $f = 6 and f_{stair} = 4$ had the best reaction to the lossy data compression.

4.3.6 Experiment Summary

Encoding K in the blue color channel with a cosine wave was shown to reduce the error by a factor of 2 in most cases. Quantization error was the same as it was with the step height encoding of K which was to be expected, but lossy JPEG encoding error was reduced drastically. The main source of spiking error in the reconstructed figures was around the boundaries and the phase jumps of the step height object. This is due to the fact that JPEG encoding favors natural images which do not have discontinuities in the color channels. Error resulting from this was also seen in the varying frequency of the fringe pattern experiment.

4.4 Summary

The three experiments performed in this chapter looked at encoding Holoimages with lossy JPEG encoding. Results from the first experiment showed that if fringe frequencies f above 6 are used, the RMSE is minimized. Results from the second experiment showed that K cannot be directly encoded into the blue color channel, even with step height variations if lossy JPEG encoding is used. Large spiking errors occurred near the phase jumps resulting in spikes on the reconstructed surfaces. The results from

Plane	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 6,\ f_{stair} = 1$	0.00%	1.44%	1.44%	1.44%	2.36%	2.63%	2.50%
$f = 6,\ f_{stair} = 2$	0.00%	1.44%	1.43%	2.04%	1.87%	2.21%	2.99%
$f = 6,\ f_{stair} = 3$	0.00%	2.04%	2.04%	1.85%	1.87%	2.21%	2.20%
$f = 6,\ f_{stair} = 4$	0.00%	2.04%	1.86%	2.04%	2.21%	2.49%	2.51%
$f = 7,\ f_{stair} = 1$	0.00%	0.85%	0.85%	1.95%	2.06%	2.60%	2.73%
$f = 7,\ f_{stair} = 2$	0.00%	0.85%	0.85%	1.66%	2.19%	1.98%	2.22%
$f = 7,\ f_{stair} = 3$	0.00%	0.84%	0.85%	0.82%	1.33%	2.07%	2.09%
$f = 7,\ f_{stair} = 4$	0.00%	0.81%	0.81%	0.84%	1.32%	2.32%	2.06%
$f = 8,\ f_{stair} = 1$	0.00%	2.17%	2.17%	2.85%	3.26%	3.12%	3.44%
$f = 8,\ f_{stair} = 2$	0.00%	1.97%	1.98%	2.16%	3.42%	3.21%	2.73%
$f = 8,\ f_{stair} = 3$	0.00%	1.98%	1.98%	1.97%	2.66%	3.44%	2.65%
$f = 8,\ f_{stair} = 4$	0.00%	1.97%	1.96%	1.97%	2.18%	2.60%	3.08%

Table 4.10 RMSE of encoding a plane with the wavy encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

Step Height	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 6,\ f_{stair} = 1$	0.93%	5.31%	5.35%	5.61%	6.35%	6.62%	7.36%
$f = 6,\ f_{stair} = 2$	0.93%	5.11%	5.19%	5.48%	5.88%	6.31%	7.06%
$f = 6,\ f_{stair} = 3$	0.93%	5.25%	5.20%	5.22%	5.80%	6.22%	6.77%
$f = 6,\ f_{stair} = 4$	0.93%	5.17%	5.07%	5.32%	5.33%	6.17%	6.44%
$f = 7,\ f_{stair} = 1$	0.89%	5.74%	5.82%	6.39%	6.36%	6.99%	7.31%
$f = 7,\ f_{stair} = 2$	0.89%	5.86%	5.91%	6.37%	6.68%	7.10%	7.44%
$f = 7,\ f_{stair} = 3$	0.89%	5.58%	5.58%	6.06%	6.16%	6.35%	6.89%
$f = 7,\ f_{stair} = 4$	0.89%	5.75%	5.76%	5.91%	6.08%	6.57%	6.64%
$f = 8,\ f_{stair} = 1$	0.95%	5.49%	5.59%	6.03%	6.23%	6.62%	7.02%
$f = 8,\ f_{stair} = 2$	0.95%	5.40%	5.48%	6.07%	6.18%	6.44%	6.81%
$f = 8,\ f_{stair} = 3$	0.95%	5.41%	5.43%	5.64%	6.31%	6.37%	6.44%
$f = 8,\ f_{stair} = 4$	0.95%	5.20%	5.51%	5.39%	6.22%	6.38%	6.84%

Table 4.11 RMSE of encoding the step height block with the wavy encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

Multiple Circles	Quantized	JPEG 100	JPEG 95	JPEG 90	JPEG 85	JPEG 80	JPEG 75
$f = 6$, $f_{stair} = 1$	0.00%	3.23%	3.60%	4.01%	4.41%	4.82%	5.52%
$f = 6$, $f_{stair} = 2$	0.00%	3.21%	3.48%	3.83%	4.27%	4.75%	5.26%
$f = 6$, $f_{stair} = 3$	0.00%	3.11%	3.22%	3.63%	3.92%	4.46%	4.95%
$f = 6$, $f_{stair} = 4$	0.00%	3.11%	3.22%	3.58%	3.87%	4.26%	4.75%
$f = 7$, $f_{stair} = 1$	0.00%	3.72%	3.94%	4.49%	5.17%	5.62%	6.07%
$f = 7$, $f_{stair} = 2$	0.00%	3.67%	3.89%	4.21%	4.73%	5.35%	5.90%
$f = 7$, $f_{stair} = 3$	0.00%	3.65%	3.77%	4.22%	4.65%	5.13%	5.78%
$f = 7$, $f_{stair} = 4$	0.00%	3.65%	3.80%	4.06%	4.61%	4.96%	5.48%
$f = 8$, $f_{stair} = 1$	0.00%	4.48%	4.70%	5.03%	5.73%	5.96%	6.46%
$f = 8$, $f_{stair} = 2$	0.00%	4.33%	4.54%	4.85%	5.38%	5.87%	6.22%
$f = 8$, $f_{stair} = 3$	0.00%	4.21%	4.35%	4.74%	5.24%	5.75%	6.10%
$f = 8$, $f_{stair} = 4$	0.00%	4.23%	4.33%	4.67%	5.20%	5.63%	6.04%

Table 4.12 RMSE of encoding the multiple spheres with the wavy encoding of K. Along the column gives the different types of encoding performed on the Holoimage, and down the rows gives increasing frequencies of encoding.

the last experiment showed that JPEG compression could be applied if the step height K was encoded with a cosine wave. This encoding fits naturally with the discrete cosine transform that is used in JPEG compression, and error on most of the figures was shown to drop in half.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

In previous chapters, the principle of the Holoimage technique has been explained, ideal results have been demonstrated, and it has been shown that the Holoimages themselves can be encoded in lossy formats. This chapter concludes the work that was done, discusses future work and how the Holoimage technique can be extended. Specifically, conclusions on what this work contributes along with its findings are concluded, and future work on different types of phase map filtering, how to extend the technique to video, and different color spaces will be explored.

5.1 Conclusion

This work has developed a new and novel way to compress 3D geometry. Specifically it makes use of structured light scanning to virtually scan 3D geometry into fringe images, which can be saved into the three color channels of a standard 2D image. Due to the parallel pixel operations that are performed to achieve the encoding, the entire encoding process can be performed on a graphics processing unit (GPU). Once in a standard 2D image format, the image can easily be saved in a lossless format such as PNG, which results in less than 0.1% RMSE. Finally, the 2D image can be decoded through triangulation, effectively reconstructing the geometry. Once again, due to the parallel pixel operations that are performed, the entire decoding process can be performed on a GPU.

Three experiments were performed on various Holoimages to see how they react

to lossy compression, and to see whether or not the pattern used by the Holoimage technique could be tailored to be more resilient to lossy image compression. The first experiment explored how changing the fringe frequency f affected lossy compression with the red and green color channel. For this experiment, the blue color channel was left in a lossless state, so that error in this channel would not compound the results. The findings suggested that higher frequencies provided a better encoding pattern, and that an optimal frequency would be in the range of [10, 12].

The second experiment looked at encoding K using a step height function, and how this encoding would react to lossy compression while varying the fringe frequency f. For this experiment the red and green color channels were kept in a lossless format so that they would not compound the error. The findings suggested that lower fringe frequencies provided a better pattern, but even with lower frequencies, this form of encoding resulted in error greater than 6%. Thus, this encoding of K does not work for lossy compression.

Finally the third experiment looked at encoding K using a cosine function, and how this encoding would react to lossy compression while varying the fringe and step frequency f *and* f_{stair}. For this experiment, the red and green color channels were kept in a lossless format. The findings showed that this format reduced the error by a factor of 2, and that upon visual inspection of the figures they looked much better. Thus, encoding K in this format works for lossy compression. Using JPEG quality level 100, a compression ratio of over $(72 : 1)$ Holoimage to OBJ can be achieved in this format. Furthermore, compressing using JPEG quality level 75, a compression ratio of over $(370 : 1)$ Holoimage to OBJ can be achieved with this encoding. Thus the Holoimage technique provides quick and efficient encoding of 3D scans at high compression ratios.

5.2 Future Work

The next major steps in this research is to more efficiently filter the phase map and move from static frames to movies. Since median filtering involves conditional statements to find the median it is not the most efficient on the GPU, and also results in some artifacts being introduced. The jump from static frames to moving frames requires the streaming of the Holoimages to the GPU as textures which is a bottleneck in the GPU. Another problem with movies formats is that they are typically in different color spaces such as the YUV color space. This requires more than just a transform back and forth, thus the color space has to be encoded directly.

5.2.1 Filtering the Phase Map

On the decoding pipeline, currently the slowest step is median filtering the phase map. This is due to the fact that the median filter involves conditional statements, resulting in divergent threads on the GPU. If another type of filtering that uses convolution could be used, this speed could be dramatically increased. Another disadvantage of median filtering is that it has the potential to flatten certain areas of the phase map. This results in small artifacts on the reconstructed geometry. Although this is unnoticeable on most scans, on the unit sphere these show up quite easily. A solution to this could be a modified median filter which looks for discrepancies and then finds figures out the correct phase jump.

5.2.2 Video

As stated before in order to jump from static frames to moving frames requires the stream of the Holoimages to the GPU as textures. If playing a previously encoded set of Holoimages, they can be transferred to the GPU as a batch of textures and then played back to produce a 3D video. This works well for applications such as replaying

a previous recording. If on the other hand the Holoimages are not previously encoded, such as those coming from a realtime 3D scanner, this technique will not work.

A technique to mitigate this is using direct memory access (DMA) with pixel buffer objects. This allows the CPU to place a texture in reserved memory for the graphics card and trigger a transfer. At this point the CPU can continue to work while the GPU performs the transfer to global GPU memory. On the GPU side, the transfer is also asynchronous of current calculations, as long as the texture is not mapped by any current operations. Thus a single frame delay can be introduced which allows the GPU to display one frame while loading another frame.

5.2.3 Video Color Spaces

Another challenge to address is the color spaces used by video codecs. Typically video codecs do not make use of the same color spaces as image codecs. An example of such a color space is the YUV color space which encodes its information in luma (Y) and chromance (UV). The reasoning behind using different colorspaces is due in part to support old displays, as YUV is backwards compatible with black and white displays. The luma (brightness) or Y component allows black and white displays to simply display the value and ignore the chromance (color) UV components. This color space is also conveniently tailored to the human eye. The human eye has a higher spatial sensitivity to luma (brightness) than chromance (color). Knowing this, bandwidth can be saved by reducing the sampling accuracy of the chromiance channels with little impact on human perception of the resulting video. An example of a codec doing this is the H.264 codec which compresses the UV components more than the Y component if more bandwidth is needed, with YUV422 compression. Although this works nicely for video being perceived by humans, it does not hold true for a Holoimage, which has been designed to have reductions in the RGB color space, but not the YUV color space. Thus a future direction in this research is how to embed the fringe patterns into the YUV

color space, and make them tolerable to compression.

5.3 Summary

In conclusion, the Holoimage technique has been shown to effectively compress 3D geometry, especially that coming from a high resolution 3D scanner. Since it only acquires geometry from a specific view, like a depth map, it cannot represent complete 3D geometry. For 3D scanners, this does not pose a problem as 3D data is only being generated from a single view. Holoimage was also shown to be tolerant of lossy compression, thus lossy image formats such as JPEG can be leveraged to hold 3D geometry. This allows the large body of 2D image compression research and 2D image infrastructure to be extended into 3D. Future research directions were explored, and directions for extending the technique to video and different color spaces have been elaborated upon.

BIBLIOGRAPHY

Chai, B.-B., Sethuraman, S., Sawhney, H. S., and Hatrack, P. (2004). Depth map compression for real-time view-based rendering. *Pattern Recognition Letters*, 25(7):755 – 766. Video Computing.

Ghiglia, D. C. and Pritt, M. D. (1998). *Two-dimensional phase unwrapping: Theory, algorithms, and software*. John Wiley and Sons, Inc, New York, NY.

Gorthi, S. and Rastogi, P. (2010). Fringe projection techniques: Whither we are? *Opt. Laser. Eng.*, 48:133–140.

Gu, X., Gortler, S. J., and Hoppe, H. (2002). Geometry images. *ACM Transactions on Graphics*, 21(3).

Gu, X., Zhang, S., Zhang, L., Huang, P., Martin, R., and Yau, S.-T. (2006). Holoimages. In *ACM Solid and Physical Modeling*, pages 129–138, UK.

Gumhold, S., Kami, Z., Isenburg, M., and Seidel, H.-P. (2005). Predictive point-cloud compression. *ACM SIGGRAPH 2005 Sketches on - SIGGRAPH '05*, page 137.

Guo, H. and Huang, P. (2008). 3-d shape measurement by use of a modified fourier transform method. In *Proc. SPIE*, volume 7066, page 70660E.

Huang, P. S. and Zhang, S. (2006). Fast three-step phase shifting algorithm. *Appl. Opt.*, 45:5086–5091.

Karpinsky, N. and Zhang, S. (2010a). Composite phase-shifting algorithm for 3-d shape compression. *Opt. Eng.*, 49(6):063604.

Karpinsky, N. and Zhang, S. (2010b). High-resolution, real-time 3-d imaging with fringe analysis. *Real Time Image Proc.* (doi:10.1007/s11554-010-0167-4).

Kauff, P., Atzpadin, N., Fehn, C., Muller, M., Schreer, O., Smolic, a., and Tanger, R. (2007). Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability. *Signal Processing: Image Communication*, 22(2):217–234.

Krishnamurthy, R., Chai, B., and Tao, H. (2002). Compression and Transmission of Depth Maps for Image-Based Rendering. *Image Processing,*, 1(c).

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., and Turner, G. a. (1996). Real-time, continuous level of detail rendering of height fields. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, pages 109–118.

McGuire, M. (2008). A fast, small-radius gpu median filter. In *ShaderX6*.

Merry, B., Marais, P., and Gain, J. (2006). Compression of Dense and Regular Point Clouds. *Computer Graphics Forum*, 25(4):709–716.

Salvi, J., Fernandez, S., Pribanic, T., and Llado, X. (2010). A state of the art in structured light patterns for surface profilometry. *Pattern Recognition*, 43(8):2666 – 2680.

Schreiber, H. and Bruning, J. H. (2007). *Optical Shop Testing*, chapter 14, pages 547–666. John Winley & Sons, 3rd edition.

Takeda, M. and Mutoh, K. (1983). Fourier transform profilometry for the automatic measurement of 3-d object shape. *Appl. Opt.*, 22:3977–3982.

Zhang, S. (2010). Recent progresses on real-time 3-d shape measurement using digital fringe projection techniques. *Opt. Laser Eng.*, 48(2):149–158.

Zhang, S. and Huang, P. S. (2006a). High-resolution real-time three-dimensional shape measurement. *Opt. Eng.*, 45(12):123601.

Zhang, S. and Huang, P. S. (2006b). High-resolution, real-time three-dimensional shape measurement. *Optical Engineering*, 45(12):123601.

Zhang, S. and Huang, P. S. (2006c). Novel method for structured light system calibration. *Optical Engineering*, 45(8):083601.

Zhang, S. and Yau, S.-T. (2008). Three-dimensional data merging using holoimage. *Opt. Eng.*, 47(3):033608. (*Cover feature*).