Models and Measurments for Multi Layered Displays

Gareth Bell
Deep Video Imaging Ltd

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
26 JUL 2006	**Final Report (Technical)**	**02-01-2003 to 12-04-2004**

4. TITLE AND SUBTITLE
Measurement and Metrics for Multi-Layer Display Technology

5a. CONTRACT NUMBER
F6256203P0133

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gareth Bell

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Deep Video Imaging, Ltd,Airport Road, Mystery Creek, RD 2,Hamilton ,NA,2005

8. PERFORMING ORGANIZATION REPORT NUMBER
AOARD-034004

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
The US Resarch Labolatory, AOARD/AFOSR, Unit 45002, APO, AP, 96337-5002

10. SPONSOR/MONITOR'S ACRONYM(S)
AOARD/AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AOARD-034004

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
?Deep Video Imaging Ltd, Airport Road RD2, Mystery Creek, Hamilton, 02021 New Zealand (Attn: Gareth Bell). The U.S. Government has a non-exclusive license rights to use, modify, reproduce, release, perform, display, or disclose these materials, and to authorize others to do so for US Government purposes only. All other rights reserved by the copyright holder.

14. ABSTRACT
Immediate concerns in multi-layered display development centre around a method to describe and measure the trade-off between Moire interference, viewed as rainbow colored bands, and image sharpness. Since displays are only wholly defined in conjunction with an observer, we need to find a map between what is perceived, what is measured and the actual design artifacts. A perceptually weighted metric describing the trade-off forms the central node. If we approach the node from the physical embodiment we need measurements and if we approach from the observer, we need to find out if what they are actually seeing is well described by the metric. A model that describes how changing the distance between a holographic diffuser and how it effects the image was verified experimentally within an order of magnitude, but would benefit from more further experimental comparison. A model was produced that predicts the Moire interference between two image layers which works well. However the metric described needs further investigation. A survey was also completed that showed that was surprising in the sense that viewers appreciate image clarity and will tolerate more Moire interference in the tradeoff described above.

15. SUBJECT TERMS
Display Technologies, Mathematical Modeling

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**80**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Abstract

Immediate concerns in multi-layered display development centre around a method to describe and measure the trade-off between Moire interference, viewed as rainbow coloured bands, and image sharpness. Since displays are only wholly defined in conjunction with an observer, we need to find a map between what is perceived, what is measured and the actual design artefacts. A perceptually weighted metric describing the trade-off forms the central node. If we approach the node from the physical embodiment we need measurements and if we approach from the observer, we need to find out if what they are actually seeing is well described by the metric. A model that describes how changing the distance between a holographic diffuser and how it effects the image was varified experimentally within an order of magnitude, but would benefit from more further experimental comparison. A model was produced that predicts the moire interference between two image layers which works well. However the metric described needs further investigation. A survey was also completed that showed that was surprizing in the sense that viewers appreciate image clarity and will tolerate more moire interference in the tradeoff described above.

Contents

Acknowledgements

Mr. Dan Evanicky for the original metric inkling.

Dr. Tae Woo Park from AOARD and Dr. Darrel Hopper from the AFRL for providing funding and their patience.

Mr. Jim Barry, Mr. Andy Hodgkinson and Mr. Gabriel Engel for their guidance and patience.

Mr. Dave Ferguson for fielding many late and often distressed phone calls.

Dr. Evan Bidder for his review of the original proposal to AOARD.

Dr. Gary Bold and Prof. John Harvey for guidance.

Dr. William Wong for his suggestions on the survey.

List of Tables

Chapter 1
Background

1.1 Multi Layered Display Technology

Multi-layered displays consist of flat panels stacked in depth and with a pre-set distance. These displays provide a method to improve information interaction and search, by providing both increased display real-estate in one view and the three dimensional cues lacking in both two dimensional displays and stereoscopic systems. Traditional graphical user interfaces make a distinction between the information being displayed, and the devices used to control it, by using depth cues such as shading and opacity to suggest the two concepts inhabit separate layers. However there exist additional cues, such as stereopsis, accommodation, convergence and parallax available on multi-layered displays which provide pre-attentive information to the viewer. This helps to decrease clutter on the display making transparent layers possible by reducing interference, as well as improving search time for required information. Thus more information can be displayed in a given area; helping the viewer maintain peripheral awareness of one data set whilst viewing another, easier switching between tasks as well as decreasing the fatigue created by turning ones head when viewing multiple monitors.

MLD architecture may be broken down into the following hierarchy:

Several display layers stacked at a preset distance from each other in conjunction with a high brightness backlight
- A display layer which is a two dimensional array of pixels
  - A sub-pixel which is composed of
    - Polarizer
    - Glass substrate
    - Electrode and driving electronics
    - Alignment layer
    - Super Twisted Nematic Liquid Crystal
    - Alignment layer
    - A red, green or blue Colour Filter/Black Matrix
    - Common Electrode
    - Glass Substrate
    - Analyzing Polarizer

The sub-pixel can be thought of light valve that controls the emitted irradiance of a red, green or blue light depending on the attached colour filter. Liquid crystal, a pseudo regular arrangement of long, polar organic molecules, retards the extraordinary ray by an amount that depends the electric field applied across it. Since the rear polarizer excludes the ordinary ray the light that exits through the front polarizer can be controlled by varying the strength of the applied electric field.

A pixel is comprised of three sub-pixels, usually red, green and blue which when

integrated by the human eye, which is usually relatively a long distance away from the screen compared to the size of each pixels, combine to form a single colour. This mechanism is often called ‘colour subtractive’, because it starts from white light which is the sum of the entire visible spectrum, and filters off the unwanted part of the spectrum with colour filters. The display is a 2D array of these pixels and includes driving electronics which takes the digital signal (usually LVDS or TTL), and addresses each sub pixel with the correct analogue drive voltage.

Chapter 2
Spatial Filtering

2.1 Objectives

To determine the effect of a diffuser acting on a target object given the:-
the distance from the display layer
the Bidirectional Transmission Distribution Function of the diffuser

2.2 Introduction

To optimize the method used to abate the perception of moire interference whilst preserving image quality in MLDs, it is important to understand how placing a spatial filter over the rear display layer effects the image formed on the retina. Some intuition may be gained by considering **??**. A light source is placed behind a small pinhole with a spatial filtering layer positioned at some distance above it. For every point on the spatial filter there is a set of rays entering that are redirected and incident on the lens and hence imaged on the retina, for example those shown in red. Conversely most rays emitted from the spatial filter do not make it to the retina, for example those shown in yellow. The illuminance and mean angle of the small bundle of rays entering at that point and output angle are parameters into the bidirectional transmission distribution function (BDTF) which determines the exit luminance. The BDTF can be thought of an intensity map for the corresponding incident and exit angles, where the spatial filter distance and viewer distance determine the path that is taken across this map, which in turn determines the distribution of the spot produced on the retina. The theory below expands this idea into a formal derivation and the experiment below seeks to verify its validity.

2.3 Theory

A simple model of the eye's lens system is assumed: a linear transform of a scene in 3D world coordinates to the 2D coordinate system of a flat retina, the same assumption as used in image processing and computer graphics. Because of variance from person to person, and for simplicity, the eye's own point spread function and any distortions introduced by the lens are ignored. The contrast sensitivity function, incorporated in the metric accounts for the qualities of the retina and subsequent processing.

To determine the effect of the spatial filter element on small features on the rear display layer consider the optical system shown in Figure: 2.

The object is partitioned into a fine grid which is mapped by the thin lens, focused to give a magnification M, to a grid on the retina

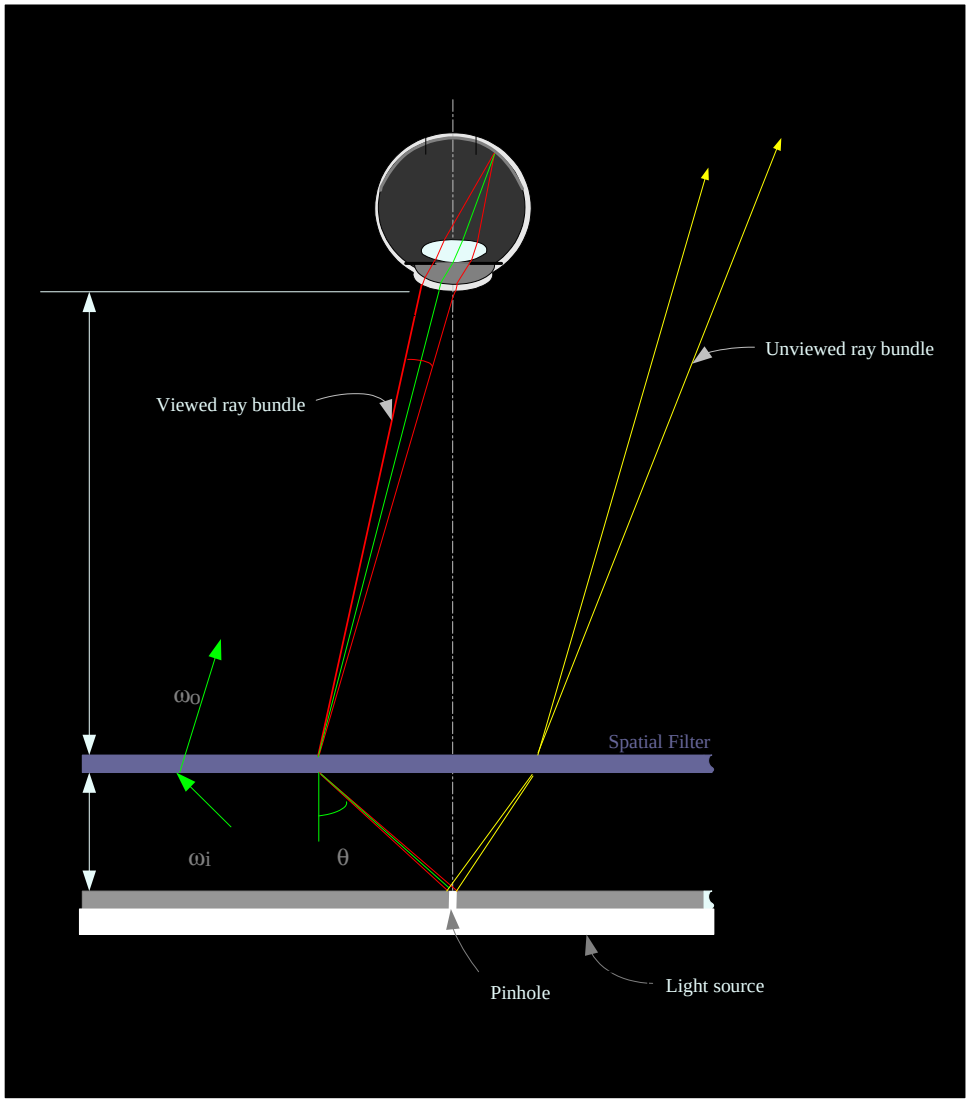


Figure 1: Overview of the spatial filtering model. The spatial filter redirects rays from the pinhole to the camera that would not arrive otherwise. The bidirectional transmission distribution function (BTDF) detirmines the luminance at that point.

Symbol	Description
x, y, z	x,y,z coordinates
Z_{OD}	Distance between the object and the diffuser
Z_{DL}	Distance along the z axis between the diffuser and the lens
Z_{LR}	Distance along z axis between lens and retina
$R_{OD}*$	Ray from object to diffuser
$R_{DL}*$	Ray form diffuser to lens
$R_{LR}*$	Ray from lens to retina
δA_D	Small area surrounding the intersection of ray we are following with diffuser plane
δA_R	Small area surrounding the intersection of ray we are following with retina plane
Ω_{OD}	Solid angle formed between point source and δA_D
Ω_{DL}	Solid angle formed between the intersection of the ray we are following and the diffuser plane
M	Magnification of the lens system
$\|\|$	Modulus of a vector
$\|\|\|$	Norm of a vector
$\rightarrow$	Direction of optic flow
ω_i	$\|R_{OD}*\|$
ω_o	$\|R_{DL}*\|$

Table 1: Notation used for Spatial Filtering Model.

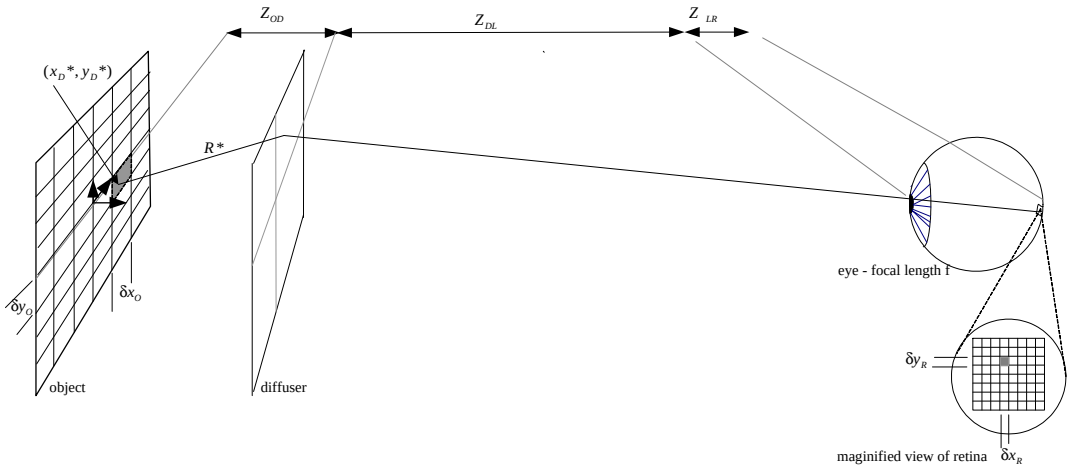


Figure 2: The notation for the spatial filtering dirivation below.

$$G_R = -M \begin{bmatrix} (x_1, y_1) & (x_2, y_1) & \cdots & (x_n, y_1) \\ (x_1, y_2) & \ddots & & (x_n, y_2) \\ \vdots & & \ddots & \vdots \\ (x_1, y_m) & (x_2, y_m) & \cdots & (x_n, y_m) \end{bmatrix} \tag{1}$$

The negative sign implies that the x coordinate has been reflected in the y axis and vice versa. A ray $R*$ from a point at (x_0*, y_0*) behind the diffuser is divided into three sections $\widetilde{R}_{OD}*$, $\widetilde{R}_{DL}*$, and $\widetilde{R}_{OD}*$. $\widetilde{R}_{DL}*$**.** The ray, whose direction is described by the inclination angles $(\vartheta_H, \vartheta_V)$, is resolved into components in the horizontal z-x plane and vertical z-y plane respectively and is redirected by the diffuser at the point

$$(x_D*, y_D*) = (Z_{OD} tan(\theta_H), Z_{OD} tan(\theta_v)) \tag{2}$$

The point $(x_D*, y_D*)$is a new object which is imaged by the thin lens. The surrounding grid element at the retina is the the x and y coordinates of the imaged point divided by the grid size at the fovea, rounded up to the nearest integer

$$(x_R*, y_R*) = (\frac{x*_D}{M}, \frac{y*_D}{M}) \tag{3}$$

$$G_R(x_R*, y_R*) = (ceil\left\{\frac{x_R*}{\delta x_R}\right\}, ceil\left\{\frac{y_R*}{\delta y_R}\right\}) \tag{4}$$

Given the irradiance of the light entering the diffuser contained within a small solid angle, the output luminance on the exit side at that point is

$$\begin{aligned} L_D(\omega_0) &= L_0(\omega_i) f_s(\omega_i \longrightarrow \omega_o) \Omega_D(\omega_i) \\ &= L_0(\omega_i) f_s(\omega_i \longrightarrow \omega_o) \frac{\delta A_D}{|R_{OD}*|^2}. \end{aligned} \tag{5}$$

Now

$$f_s = \frac{dL_0(\omega_0)}{L_i(\omega_i) d\Omega(\omega_i)} \tag{6}$$

is the BDTF for the diffuser element where

$$\begin{aligned} \omega_i &= \|R_{OD}\| \\ \omega_o &= \|R_{DL}\|. \end{aligned} \tag{7}$$

The illuminance at the lens is

$$\begin{aligned} E_L &= L_D(\theta_H, \theta_V) \delta\Omega_{DL} \\ &= L_D(\theta_H, \theta_V) \frac{A_{lens}}{|R_{DL}*|^2} \end{aligned} \tag{8}$$

and the flux through the lens is

$$\begin{aligned} \phi_{lens} &= E_L A_{lens} \\ &= L_D(\theta_H, \theta_V) \frac{A_{lens}^2}{|R_{DL*}|^2} \end{aligned} \tag{9}$$

The illuminance imaged at each grid area on the retina by the lens, the stimulus at that area, is the flux through the lens and divide it by the area of the corresponding grid element

$$E_v \Big|_{\substack{x_R \\ y_R}} \simeq \frac{\phi_{lens}}{\delta A_R} \tag{10}$$

So

$$E_v \Big|_{\substack{x_R \\ y_R}} \simeq \frac{f_s(\omega_i \rightharpoonup \omega_o) L_O(\theta_H, \theta_V) A_{pupil}^2}{(\frac{x_R^2}{M^2} + \frac{y_R^2}{M^2} + Z_{OD}^2)(\frac{x_R^2}{M^2} + \frac{y_R^2}{M^2} + Z_{DL}^2)} \frac{\partial A_O}{\partial A_R} \tag{11}$$

However the result should be independent of the grid and so dissolve the ratio on the right hand side. The

$$\lim_{\partial A_R \to 0} \frac{\partial A_D}{\partial A_R} = \frac{\partial A_O}{\partial A_R} \tag{12}$$

for

$$\partial A_F = f(\partial A_O) \tag{13}$$

and since

$$\begin{aligned} \partial A_R &= \partial x_R \partial y_R \\ &= M \partial x_0 M \partial y_0 \\ &= M^2 \partial A_0 \end{aligned} \tag{14}$$

which gives

$$\frac{\partial A_R}{\partial A_0} = M^2 \tag{15}$$

and so

$$E_v \Big|_{\substack{x_R \\ y_R}} \simeq \frac{f_s(\omega_i \rightharpoonup \omega_o) L_O(\theta_H, \theta_V) A_{pupil}^2}{(\frac{x_R^2}{M^2} + \frac{y_R^2}{M^2} + Z_{OD}^2)(\frac{x_R^2}{M^2} + \frac{y_R^2}{M^2} + Z_{DL}^2) M^2}$$

where

$$\begin{aligned} \theta_H &= \arctan(\frac{x_R}{MZ_{OD}}) \\ \theta_V &= \arctan(\frac{y_R}{MZ_{OD}}) \end{aligned} \tag{16}$$

2.4 Experiments

There are two parts to the following experiment, the first is to measure the bidirectional scattering distribution function and to use the model outlined in the theory to predict the contrast ratio of a model target. The second part measures the contrast ratio of the target directly with a camera. The results of the two approaches are then compared. In the following section we describe the approach as a whole and examine the results for the 60°x10° elliptical diffuser measured in the 10° direction, denoted 60°x10°(10°).

2.4.1 Bidirectional Scattering Transfer Distribution Function Measurement

2.4.1.1 Method

Light Shaping Diffuser technology has proved successful in commercial Multi Layered Displays because it was found to allow fine control of the amount of the "image clarity"[1] of the rear display layer, largely due to its custom scattering properties. Five samples were sent to the National Institute of Standards and Technology [2] for BTDF measurements for comparison with the point spread function scattering measurements made in the next section. They were cut from the original 52 mm x 52 mm square samples received from POC^{TM} and had minor surface blemishes.

The data received from NIST are reported as Bidirectional Transmittance Distribution Function, defined by

$$BTDF = \frac{d\Phi_s}{d\Omega \Phi_i \cos\theta_t}$$

[1] This was the in-house term for the quality of the rear display, the improvment of which has been crucuial to the commercial viability of the product. It was important to realise the "image clarity" actually depended on both the area contrast of the image and the amount of "blur" being applied.

[2] Measurments made by:
Thomas A. Germer
Optical Technology Division
National Institute of Standards and Technology
100 Bureau Drive Stop 8442
Gaithersburg, MD 20899-8442
United States

Sample	Type	Size
1	0.2°x40°FWHM[3], 10 mil polyester substrate	13 mm x 36 mm
2	80°FWHM, 10 mil polycarbonate substrate	13 mm x 52 mm
3	60°x10°FWHM, 20 mil polycarbonate substrate	13 mm x 52 mm
4	15°x5°FWHM, 10 mil polycarbonate substrate	13 mm x 36 mm
5	30°FWHM, 10 mil polycarbonate substrate	52mm x 52mm

Table 2: Optical Sample Description.

where
$d\Phi_s$ is the measured scattered power.
$d\Omega$ is the solid angle collected by the detector $d\Omega = \frac{A}{R^2}$
where
A is area of detector
R is distance of detector from sample.
Φ_i is the total incident power
θ_t is the scattering angle, measured with respect to the surface normal.

The BTDF is the fractional power scattered per unit projected solid angle, so for a Lambertian scatterer the BTDF is constant. The factor of $\cos\theta$ was noted in the comparison below.

The following is quoted from the NIST report:
"For each incident angle, scattering angle, wavelength, polarization, and sample orientation, measurements were performed at six locations on the sample. The reported values represent the averages and standard deviations of the mean of the measurements. The observed statistical variation in the data results from laser speckle. No systematic uncertainties, which are expected to be less than 2 % in magnitude, are included in the results.
An issue that arose during measurements:
Since these measurements were performed on transmitting samples, a significant amount of stray light arises from the end of the sample and from the surface mount. Below are photographs of Sample #1 illuminated from behind at 45°, with and without the room lights on. The detector views only a small region near the illuminated spot, but some light can be captured from other reflections. This effect will be strongest at large viewing angles and large incident angles."

2.4.1.2 Results

Figure: 5 shows the results for the 60°x10° holographic diffuser (elliptical) when measured in the 10° direction. The distribution is advertised as Gaussian in the sales literature with a full width at half maximum of 10°, and a negative parabola would be expected in the log plot above. This is true until the exit angle (along the horizontal axis) is larger than 20° due to the stray light at large angles. There is a clear difference between the samples when measured in the horizontal and vertical directions, and the full width at half maximum for all samples corresponds with that advertised. Please see Appendix ??

Item Description	Serial Number
Dolan Jenner Fiber optic illuminator	027026
Dolan Jenner Flat area light	
Basler Digital Camera	10920022535
Boom Stand	
Infinitube	
Mitutoyo 5x Infinity Corrected Objective	378-802-2
Sinusoidal Target (transmission)	51
Micromo Linear Stage controller	16249
Micromo Linear Stage controller	
National Aperture Linear Translation Stage	7340
National Aperture Linear Translation Stage	
TMC Optical Breadboard	75SSC-113-12
Custom Bracket I	
Custom Bracket II	

Table 3: Equipment list and serial numbers where avaliable

for the scans of the remaining samples.

2.4.2 Point Spread Function Measurement

2.4.2.1 Equipment Setup

Please see Figure: 7, Figure: 6 for the images and schematics of the experimental setup respectively. A list of equipment and serial numbers where available are given in Table: 3

2.4.2.2 Method

It took a large amount of effort to get the data acquisition system working correctly. The first challenge was controlling the linear stages via the serial port using MATLAB®. There were two problems here:

The first was that the instructions supplied by National Aperture^{TM} with the linear motion development system, and the example software seemed to indicate that one controlled the stage using quadrature counts, the basic idea being that a given number of counts would move the stage a given amount of distance. One assumes the distance moved should be the same each time and directly proportional to the number of counts. This didn't prove to be the case, and after much exasperation the stage was used in velocity mode, that is the controller that was connected to the serial port regulated the velocity accurately, and the computer was used to track the time. This was calibrated, as outlined in appendix ??? using the computer and the camera and target. The target was placed on a glass stage, which was in turn attached to the controller stage. The stage was sent into motion for two seconds, using the function *pause* to hold command execution for this period of time, and its initial and final position were recorded in terms of pixels on

the display.

The second issue was the problematic bundled serial port control functions. Eventually the user contributed library *cport* was used to send and receive data and the problems disappeared.

The final challenge for positional control was developing the inhouse *stagehome* function. This started the stage moving and then using a while loop "listened" for an external event , actuated by one of the controllers pins going to ground which occurred when either the:-

1. Bracket attached to the stage hit either a spring loaded panel pin[4]
2. The copper pad attached to the glass stage contacted the copper pad attached to the sample as shown in Figure: 6.

The controller then reversed its direction for a couple of seconds and approached the stage more slowly the second time, and listened for the external event again and stopped.

It took three goes to do the data collection well. The first attempt had a list of target and sample positions which was randomized, and the both the target stage and the position stage were moved to home between each run. This took a long time, provided no feedback on the data that was being gathered and the scans were too far apart to be of any use.

In the first repeat the data was collected through a graphical user interface (GUI) programmed in MATLAB® that controlled the camera, the linear stages and reported the graphed calculated contrast ratio and a representation of the image back to the user in real time. The user could record a single shot and move both the target and the stage. The GUI handled storing the contrast ratio, time stamp, image, current target in a data structure. The contrast ratio at a particular sample position was calculated by using the function *findpeaks* to return mean maximum and minimum values of the imaged sine function taken from an array containing the average of the columns of the luminance values of the image. Unfortunately on this attempt the position in the output data structure was not recorded.

In the final attempt the GUI was retained and the mean contrast ratio over four images was recorded before moving the sample. A scan button was included to move the sample across the distance range which could be changed depending on anticipated results, as there was no use collecting data if the contrast ratio was near unity. The average contrast ratio was displayed after gathering each data point which provided valuable feedback as to the status of the experiment. This approach would be recommended for any future work as it provides lots of feedback as the experiment is progressing, but is not too arduous.

A photograph of the equipment setup is shown in Figure: 6 and a diagram of the sample mounting and home positioning in Figure: 7.

Before the sample was changed a control run was usually done, to ensure the camera hadn’t moved out of focus or for other defects as well as providing useful reference for the analysis below. The target stage was leveled using the dial gauge to 0.02 mm in 5mm

[4] For those not familier with the cabinet making industry this is a small nail about 6-7mm in length. This was referred to as a "poor man’s contact switch" by the electrical engineers in the building, but performed well throughout all experiments.

of movement. Ambient light was excluded by placing a black cloth over the equipment whilst running.

2.4.2.3 Results

The results shown in Figure: 9 confirm what has been known "by eye". The contrast ratio monotonically decreases as the diffuser is moved further from the target. The contrast ratio of the target without any intervening diffuser decreases as the spatial frequency increases, so it is expected that the contrast ratio is different for each target when the diffuser is very close. It is also noted that the curves are parallel for each target as would be expected. However the results are disappointing in the sense that one would expect a clearer trend when comparing between samples: the graphs are trailing off quicker in general as the as the full width half maximum angle of the diffuser sample increases but this should also occur monotonically. Both scans of the 60°x10° sample, and the scan in the 40° direction of the 40°x0.2° sample are anomalous in this respect. These samples were checked manually with a laser to ensure there was no labeling error. The remaining possibility is the stage not homing to the correct position, or the camera was bumped slightly out of focus.

2.5 Analysis

The analysis was performed using seven custom coded MATLAB functions:

1. CompareResults
2. calculateFilter
3. btdfFilter
4. createModelTarget
5. convertData
6. findPeaks
7. processing

The process was to first calculate the model filter that expressed what the point spread function would be for a given spatial filter and the camera at a given distance.

First the $\ln(BTDF)$ was plotted against the incident angle (α) so that the peaks were centered. The BTDF data was converted into ***.mat** files suitable for use by MATLAB® by using the *process* function which extracted the average BTDF files and the θ_t angle and the results were averaged from the red, green and blue laser output. A two degree polynomial was fitted to the $\ln(BTDF)$ curve, where the values that were less than the full width at half maximum were omitted. The curves were then interpolated between using the *griddata* function and a mesh plot over a range of incident and exit angles as shown in **??** was produced to ensure this was working correctly. According to the model the resulting point spread function at the sensor may be obtained by following a path on this graph that is defined by the entrance and exit angles in and out of the diffuser. These

were calculated and sent, along with the polynomial coefficients to the function *btdfFilter* to calculate the point spread function, by interpolating between the path at each point to form the BTDF for the subset of points required. Lastly the point spread function(PSF) was calculated using the formula specified in the model and the results passed back as an array as a function of position. Several calculated filters are shown in 2.4.2.3. These show the correct behavior in that the irradiance drops as the distance increases and the distribution wider.

To compare the results obtained with those that were expected from the BTDF measurements a target was generated to be within the contrast ratio bounds, and with a ripple of the same spatial frequency of a measured control target. This was then filtered using the *imfilter* function, and resulting contrast ratio was calculated.

The measured contrast ratio was calculated by taking the average luminance of the columns of the Y[5] component of the image. The results were then compared by plotting the measured and expected contrast ratio functions side by side. The graphs are shown in Figure: 8.

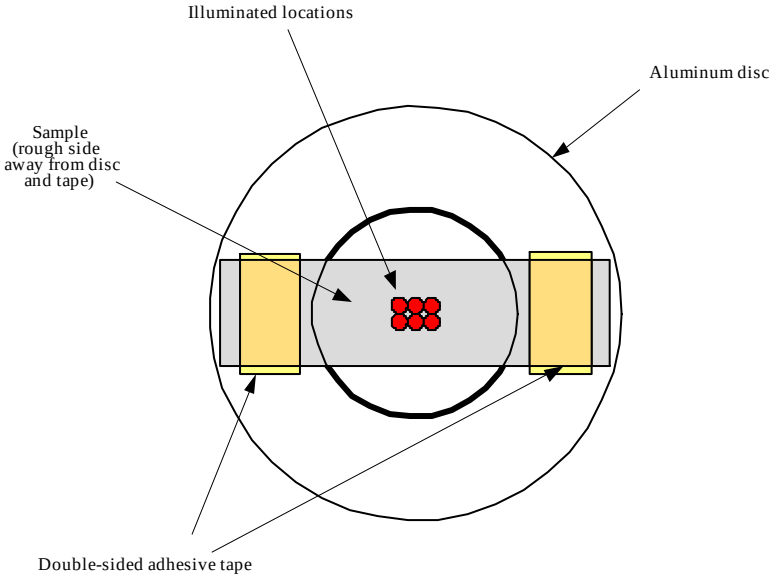
2.6 Discussion

The spatial filtering experiments were definitely the most difficult and time consuming of the sections work outlined for this author. It took a much more effort than necessary to get the linear stages working correctly, which could have been saved through good documentation. It is little disappointing that the author didn't pay more attention when dealing with the 15° x 5° diffuser sample which was lost prior to doing the final round of experiments.

2.4.2.3 shows the calculated image filters from the data from NIST and shows the correct behavior. The spot size increases and the contrast decreases as the image diffuser distance increases which is exactly as expected.

The contrast decreases as the diffuser display distance increases which was as expected, however it does not decrease monotonically as the FWHM of the diffuser increases. This is cause for suspicion that image was out of focus for the 60° x10° sample as it does not follow this trend. Figure: 8 works reasonably well at relatively large target distances, where the expected contrast ratio decreases as the diffuser distance increases and spatial frequency of the target increases. The reason for the for the rounding of the measured is the mechanical slack in the system, which means that the actual sample does not begin moving for some distance after the stage begins to move. Also the measured contrast ratio drops more abruptly at small distances, where as the expected tails off, probably because of a combination of stray light, the camera noise floor and the lower limit to the quantisation.

[5] The camera outputs results in the Y,u,v format where Y corrisponds to the luminance, and the gain of the camera is known to be linear from the manual. This is also confirmed (up to a limit) in Appendix ??.



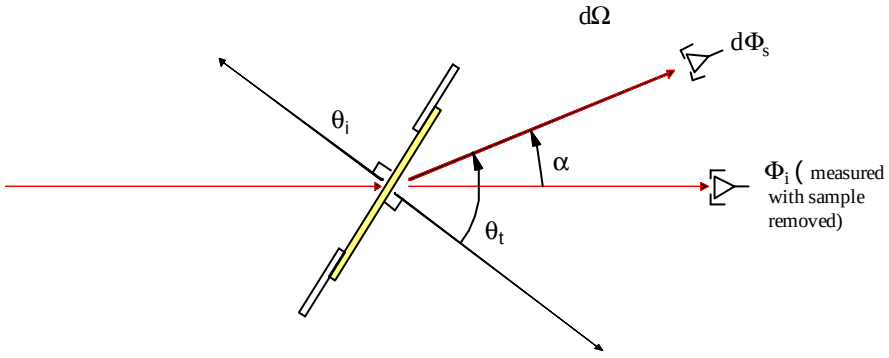


Figure 3: Measurment setup and geometry from NIST report

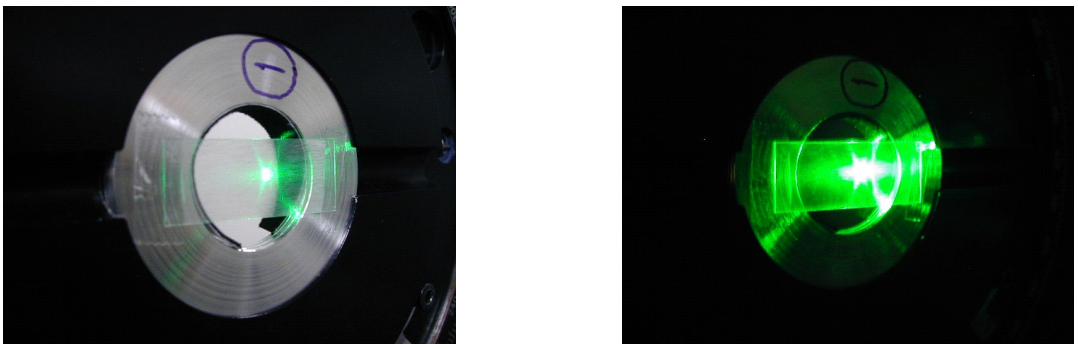


Figure 4: The light scattering problem noted above. One should not be able to see the sample mounting glass when imaged in a dark room.

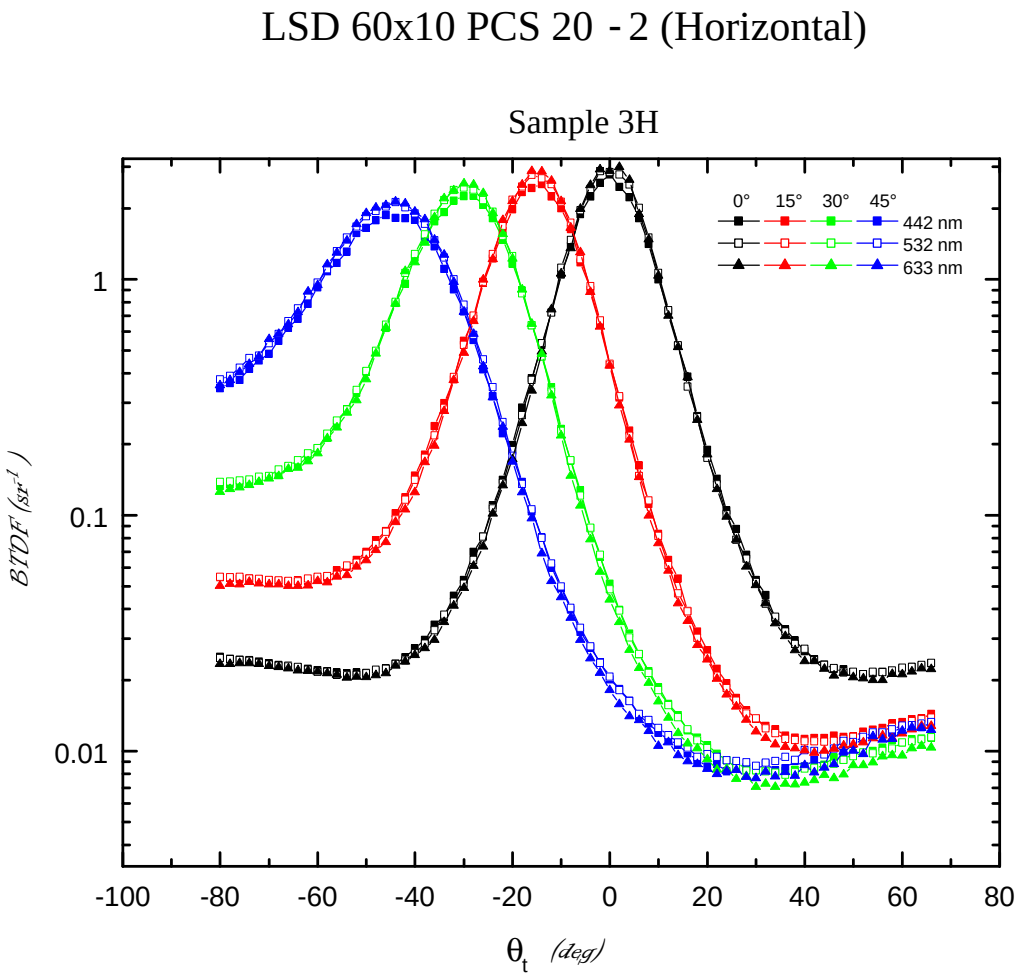


Figure 5: BTDF results for 60°x10°(10°) holographic diffuser. Note the wide wings on the bottom due to the scattering problem, which were ommitted during the model fit.

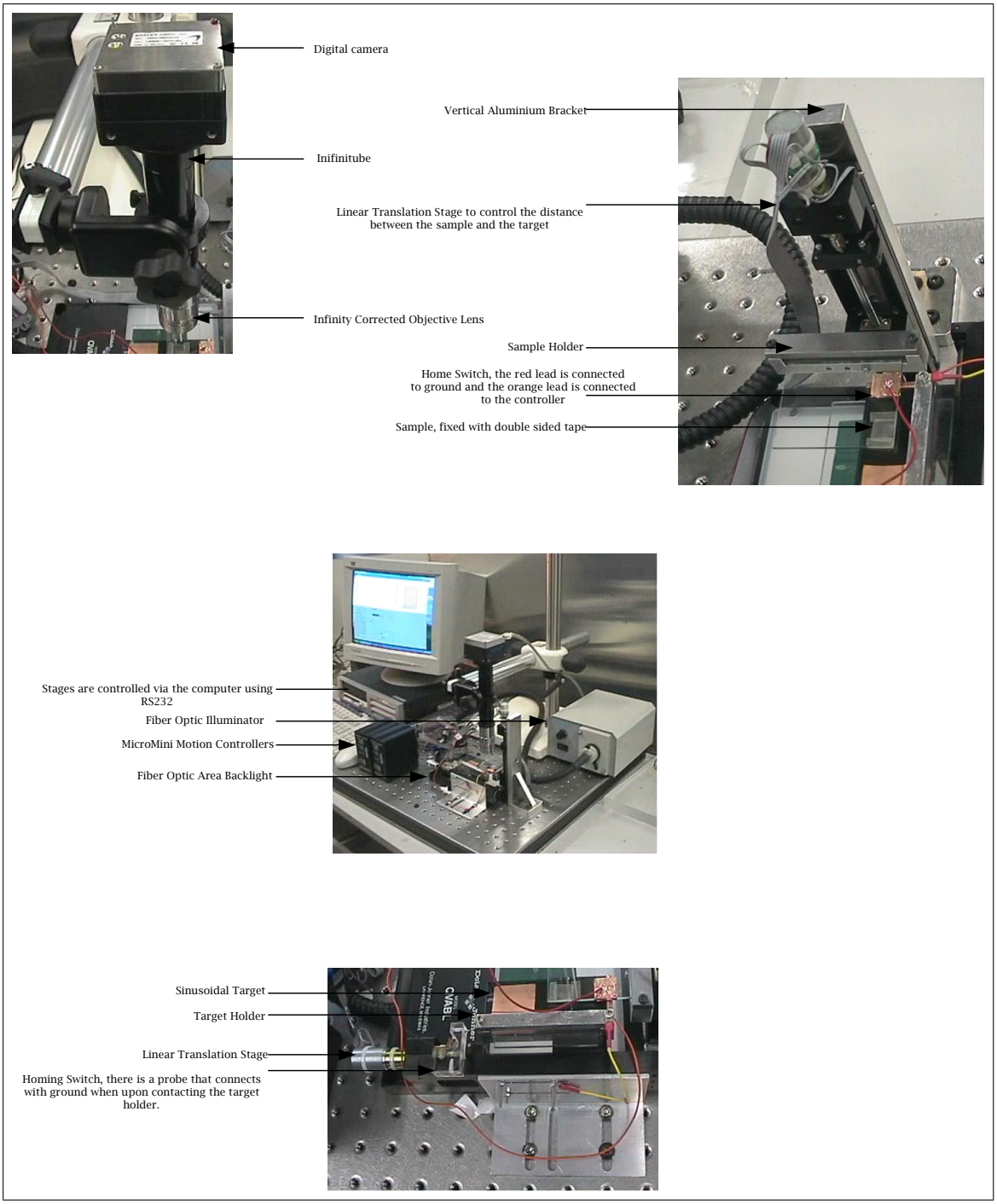


Figure 6: Experimental setup.

Section 2.6 Discussion

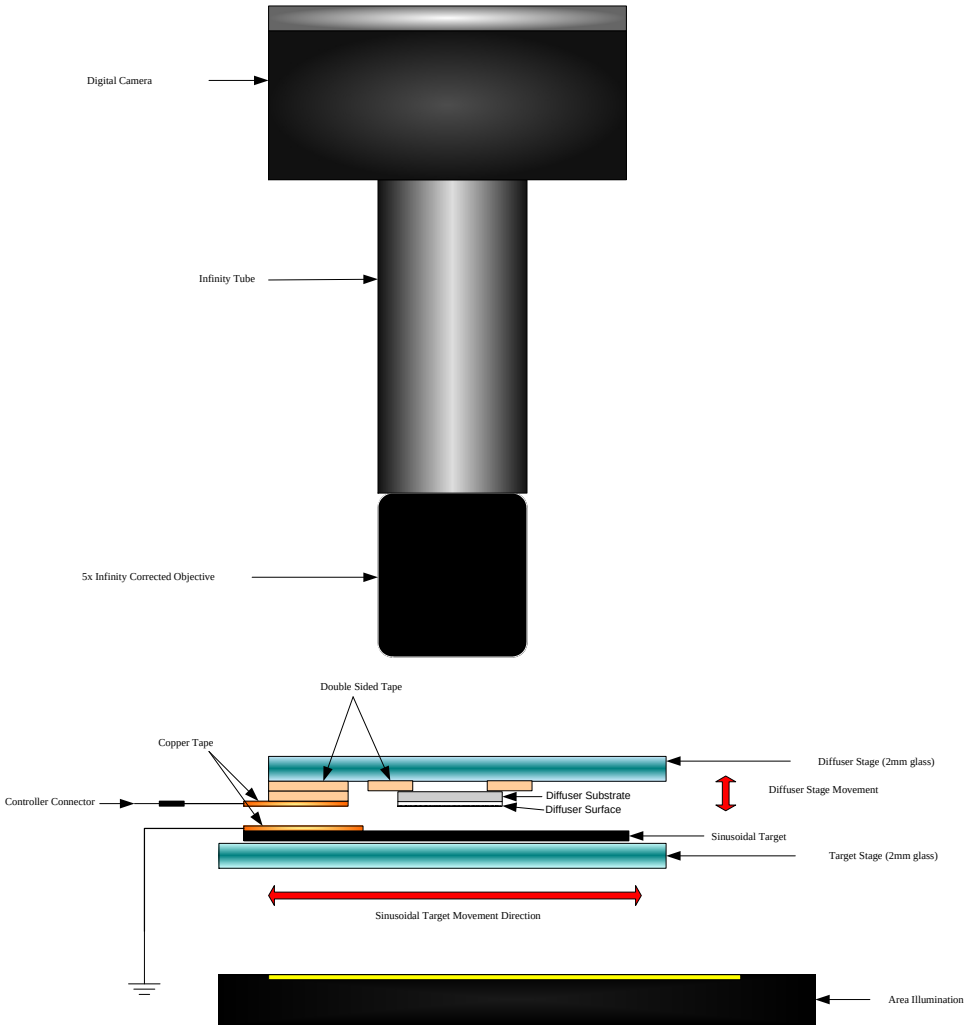
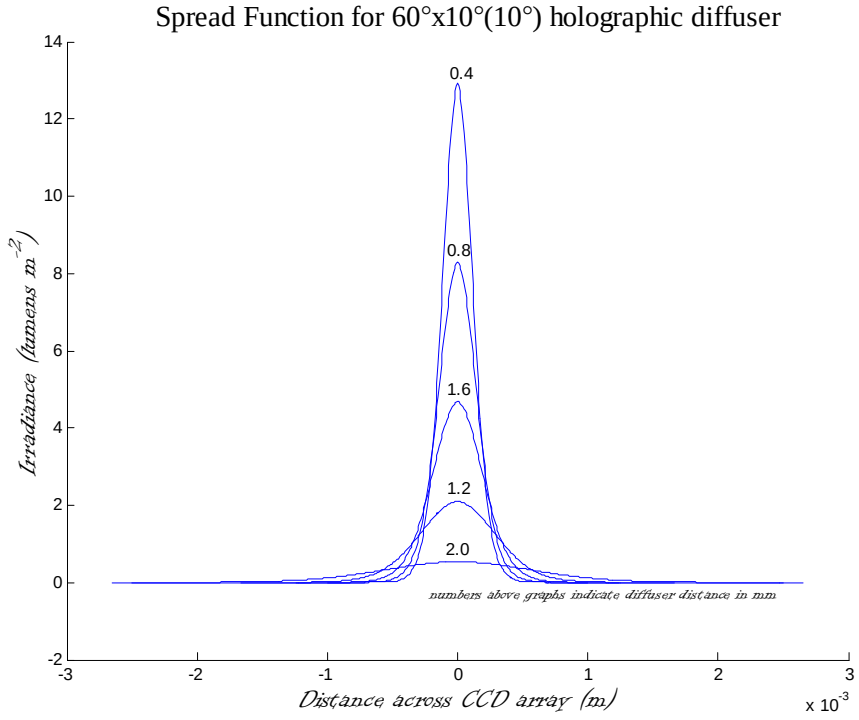


Figure 7: Schematic view of the experiemental setup showing the optical sample mounting and homing mechninisim.


Spread Function for 60°x10°(10°) holographic diffuser
Irradiance (lumens m-2)
Distance across CCD array (m)
x 10-3
0.4
0.8
1.6
1.2
2.0
numbers above graphs indicate diffuser distance in mm
14
12
10
8
6
4
2
0
-2
-3
-2
-1
0
1
2
3

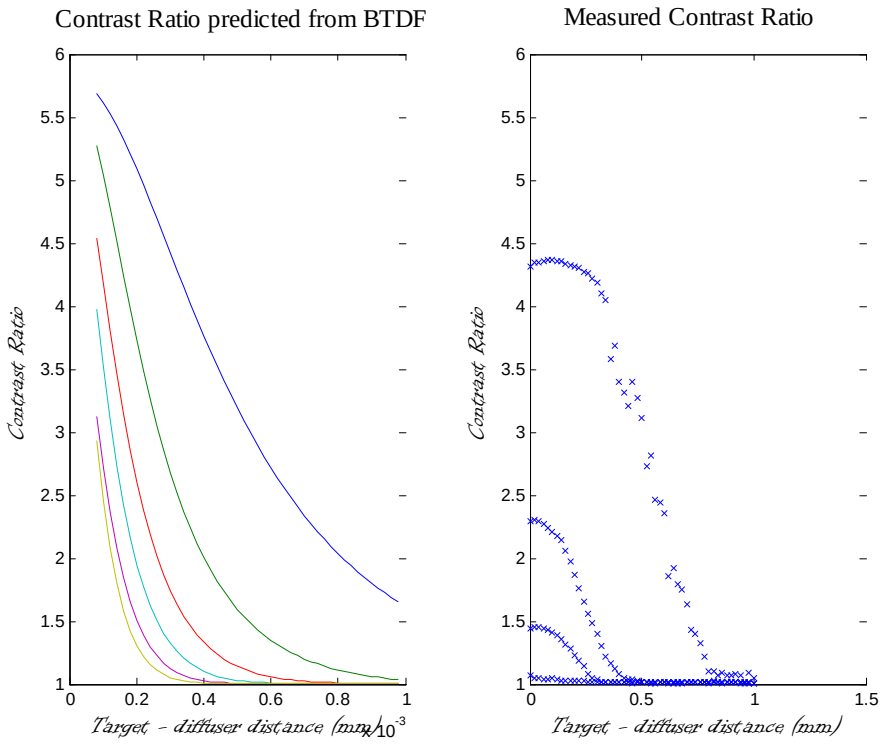


Figure 8: The measured (left) and expected (right) contrast ratios

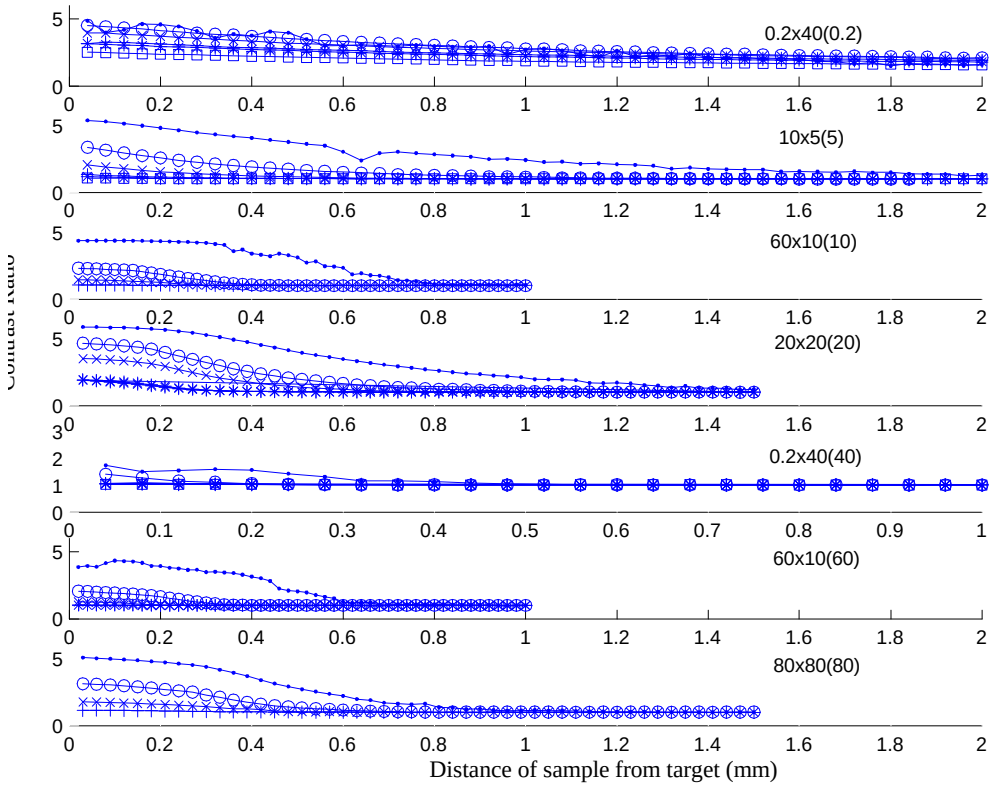


Figure 9: A comparitive plot of all the contrast ratio measurments.

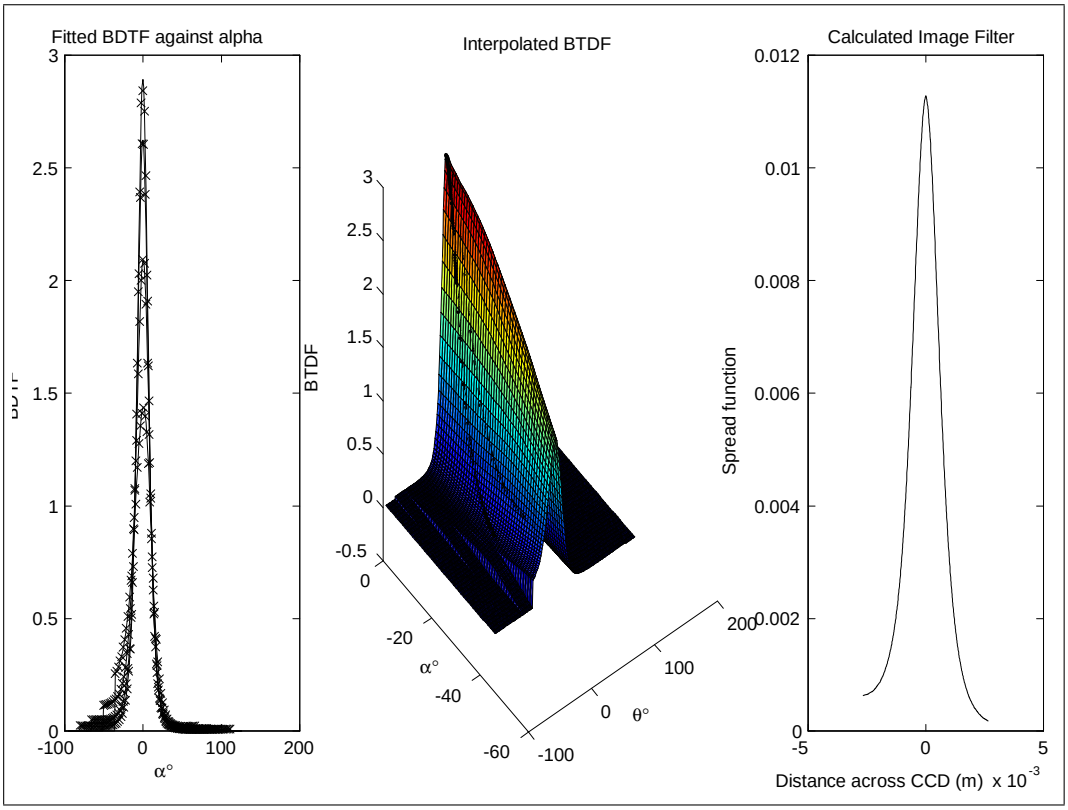


Figure 10: The analysis process: (a) The BTDF functions were plotted against the deviation from incident angle α and an exponential fitted to each curve (b) The curves were interpolated and the incident-exitance path is shown plotted for a target-spatial filter distance of 1mm (c) the curve shown as the predicted irradiance at the CCD array of the camera.

Chapter 3
Moire Interference

3.1 Aims

1. To develop a numerical model to predict the saliency of moire interference in a Multi-layered Displays
2. To verify this model by experiment

3.2 Introduction

Moire interference occurs whenever two regular patterns of slightly different spatial frequency overlap. It is an example of the beating phenomenon, and Figure: 11 provides some insight into the mechanism in multi-layered displays. There are two image layers composed of coloured filters. The planes are projected onto the retina in perspective and because the projection of the rear plane is slightly smaller than the projection of the front plane the layers have a slightly different spatial frequency. This means that a given type of filter on one layer will move in and out of phase with the corresponding filter on the next layer. When a light ray passes through filters that are the same, the saturation increases and the luminance drops slightly which is constructive interference. When light passes through two filters that are different, for example red on the first layer and blue on the second then the colour becomes less saturated and the luminance drops by about 80% resulting in destructive interference.

3.3 Theory and Software Model

To begin the displays are separated into the three different types of colour filters and the resulting nine interactions are considered separately and then added in the CIE1931 colour space. The combined transmission spectrums of the sub-pixels are described by

$$\begin{bmatrix} R_1(\lambda)R_2(\lambda) & R_1(\lambda)G_2(\lambda) & R_1(\lambda)B_2(\lambda) \\ G_1(\lambda)R_2(\lambda) & G_1(\lambda)G_2(\lambda) & G_1(\lambda)B_2(\lambda) \\ B_1(\lambda)R_2(\lambda) & B_1(\lambda)G_2(\lambda) & B_1(\lambda)B_2(\lambda) \end{bmatrix} \tag{17}$$

whose diagonal shows the brightest combinations, because the frequency spectrums for the same or similar filters are highly correlated. This can be simplified by measuring the CIE1931 coordinates, which are L, x, y for each combination and converting to X, Y, Z coordinates and so a $3 \times 3 \times 3$ array

$$[R_1 G_1 B_1] \times [R_2 G_2 B_2] \times [XYZ] \tag{18}$$

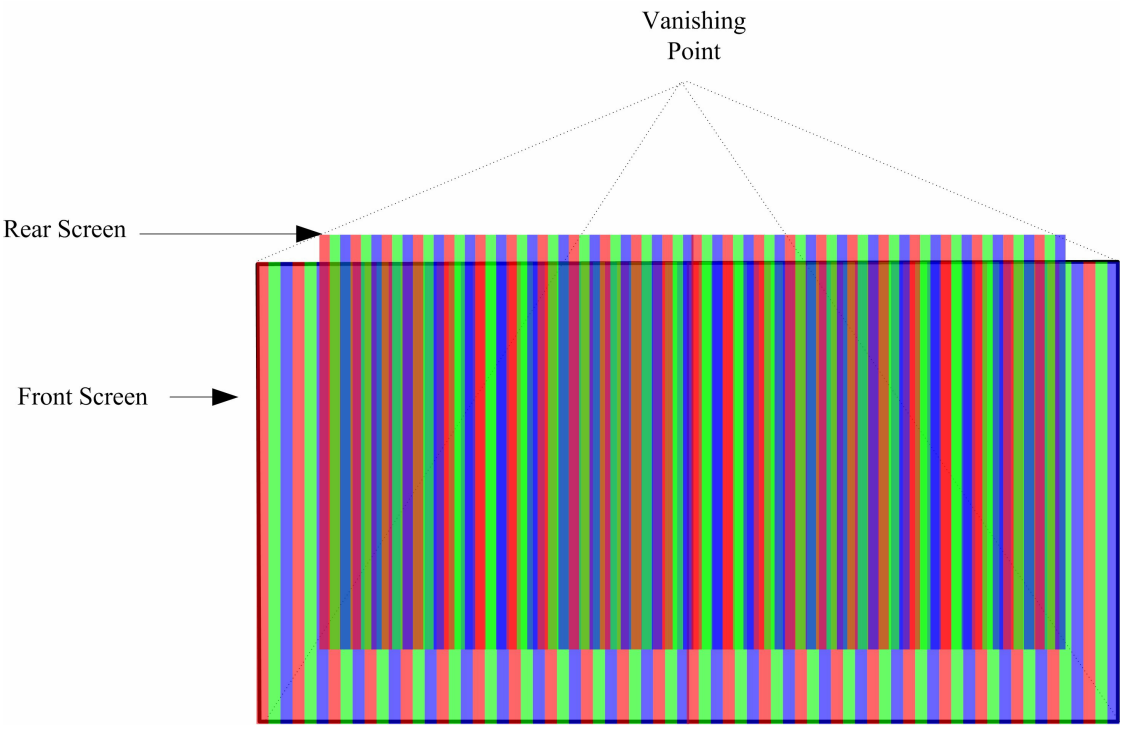


Figure 11: Moire interference in multi-layered displays. One observes dark and light bands if the pixels are in the same order on both layers.

fully describes the colour interactions fully as far as the visual system is concerned. In practice each combination can be measured by removing two sets of subpixels (by making them black) on each layer and recording the tristimulus values. To model the colour interaction of the interference between the layers, the colour on the rear layer becomes the combined colour and luminance of the two layers, and the front layer is just a suitably scaled black mask with apertures. The luminance distribution of the rear layer, for each different pixel type, is convolved with the normalised version of the digital filter calculated in section ?? and multiplied point wise by each of the three masks to determine the interaction for all combinations.

The sum of all the luminance of all combinations is then taken to predict the final luminance distribution of the interference. These results can also be converted to RGB coordinates using a 3x3 matrix transform and displayed on the screen to ensure the qualitative predictions of the model are correct.

The code below shows this process

```
function Moire = createMoire(R,F,DD,DE,CM,D)
%————————————createMoire(R,F,DD,DE,CM)————————————————%
%Adds the colours of two images. If the images are of a different size %
%then we take the intersection of the two starting from the top right
%hand corner
%————————————————————-Variables————————————————-%
% R - the rear image in RGB co-ordinates
% F - is the front image in RGB co-ordinates
% DD - is the distance between the image layers
% DE - is the distance from the front most display to the eye
% CM - is the matrix that describes how the colours interact
%_____________________________________________________________________________%
%scale the displays

D1 = DD+DE;
D2 = DE;
FS = imresize(F,(D2/D1));
S_FS=size(FS);
x=1:S_FS(1);
y=1:S_FS(2);

%multiply each layer pointwise - we are trying to find where in space each layer interacts
%and at present this is an RGB before hand, but we seperate each of the layers out and
%and interact the red layer of the front screen with the red layer of the rear screen
%for example until we have all nine combinations. Each one of these combinations
%is the luminance distribution for that colour combination

A1=ones(size(FS,1),size(FS,2),3,3,3);

for k=1:3
```

```
 for m=1:3
 A1(:,:,k,m,1) = R(x,y,k) .*FS(:,:,m) * CM(k,m,1);
 end
end

%Now for each combination k,l we need to assign CIE co-ordinates
for k=1:3
 for m=1:3
 A1(:,:,k,m,2) = ones(size(A1,1),size(A1,2))*CM(k,m,2);
 A1(:,:,k,m,3) = ones(size(A1,1),size(A1,2))*CM(k,m,3);
 end
end

%change to tristimulus values so that we may take the artihmetic sum
for k=1:3
 for m=1:3
 A2(:,:,k,m,:) = xyL2XYZ(A1(:,:,k,m,:));
 end
end

%Go through and add into a single image
A4 = sum(A2,3);
A5 = five2threedims(sum(A4,4));
%now change to rgb co-ordinates
A6 = cie2rgb(A5);

%return the CIE tristimulus values
Moire=A5;
```

3.3.1 Square Root Integral Difference Metric

We have developed a metric to quantify both the theoretical and measured Moire interfarence, with the aim of providing a measure of the its saliency as it relates to the human visual system. Barten's square root integral metric (SQRI) is a well established method to measure image quality and provides a useful starting point. 7 JND's of the SQRI indicates that a signinficant amount of distortion is present as far as the viewer is concerned, and 1 presents distoriton that would be noticed by only half of the population. Two adaptations need to be made for our purposes. Firstly we are not interested in frequency components that have been modulated, but rather the difference between the interference and the ideal flat field. Secondly the original metric was designed to act on the modulation transfer function of a system that transfers an object to an image. So an artificial "object" in the form impulse is chosen as a reference. In discreet terms we consider both a flat field and the measured or calculated Moire as response of the system

to a single pixel and take the difference of the two. The square root integral is

$$SQRI = \frac{1}{\ln 2} \int \sqrt{\frac{M(u)}{m_t(u)}} \frac{du}{u} \tag{19}$$

where
u is the spatial frequency in cycles per degree
$u_{\min}$is the lowest displayed spatial frequency
$u_{\max}$is the lowest displayed spatial frequency
$M(u)$is the MTF of the imaging system
$m_t(u)$is the modulation threshold of the eye
and

$$S(u) = \frac{1}{m_t(u)} = \frac{5200e^{-0.0016u^2(1+100/L)^{0.08}}}{\sqrt{(1+\frac{144}{X_o^2}+0.64u^2)(\frac{63}{L^{0.83}}+\frac{1}{1-e^{-0.02u^2}})}} \tag{20}$$

u is the spatial frequency in cycles per degree
L is the average luminance in cd/m^2
X_o is the angular object size in degrees
and the corrisponding code is

```
%————SQRI(Luminance,testImage,referenceImage_0,X,Y)————-%
%gareth.bell@deepvideo.com
%
%Calculate the square root integral metric for a viewing distance of
%570mm given a reference and a distorted image.
%————————————-Variables————————————-%
%
% testImage - The image to be compared
% referenceImage - The reference image
% X - The size of the image in the x direction in mm
% Y - The size of the image in the y direction in mm
%————————————-Notes————————————-%
%See Quality Aspects of Computer based Video Services for details
%Major revision and simplification given Peter Barten's presentation pg 27
%_____%

VIEWING_DISTANCE = 570; %Standard viewing distance

% Find the size of the image
[nPixels, mPixels] = size(referenceImage);

%Convert the input size into degrees assuming viewing distance of 570mm
thetaX = atan(X / VIEWING_DISTANCE) * 180/ pi;
thetaY = atan(Y / VIEWING_DISTANCE) * 180/ pi;

%Calculate the size of the increments
```

```
deltaU = 1 / thetaX;
deltaV = 1 / thetaY;

%Do the fast fourior transforms of both images
F_testImage = fft2(testImage);
F_referenceImage = fft2(referenceImage);

%Find the modulation transfer function
mtf = abs(F_testImage ./ F_referenceImage);

%We need to create a grid of u and v values
[U, V] = meshgrid( [deltaU:deltaU:(mPixels * deltaU)] , [deltaV:deltaV:(nPixels * deltaV)] );
%Calculate the frequency into the 1D Contrast sensitivity function
frequency = sqrt( U .^2 + V .^2 );

%Calculate the contrast sensitivity function
localCsf = csf(frequency, Luminance, thetaX, thetaY);

%This needs to be integrated over a log grid
logGrid = 1 ./ ( U .^2 + V .^2 );

%Calculate the final integral
I = 1./(2 * pi * log(2) )...
 * sum( sum( sqrt(mtf .* localCsf) .* logGrid * deltaU * deltaV ) );
```

And the adapted difference metric in two dimensions is

$$\Delta SQRI = \frac{1}{2\pi \ln 2}\left[\sqrt{\frac{M_{test}}{m_t(u)}} - \sqrt{\frac{M_0}{m_t(u)}}\right]\frac{dudv}{\sqrt{u^2+v^2}} \tag{21}$$

where

$$M_{test} = \left|\frac{\mathcal{F}(I_{test}/\overline{I_{test}})}{\mathcal{F}(\sigma)}\right| \tag{22}$$

and

$$M_0 = \left|\frac{\mathcal{F}(1)}{\mathcal{F}(\sigma)}\right| \tag{23}$$

```
%————————————out = moire(testImage,nullImage,X,Y)————————————%
%calculates the square root integral of the moire interfarence present
%————————————————————————-Variables————————————————-%
% testImage - rgb image with moire interfarence present
% nullImage - image using same camera on flat field of same luminance
% X - size of the image in the x direction in millimeters
% Y - size of the image in the y direction in millimeters

testYuv = mean(testImage);
```

```
nullYuv = mean(nullImage);

test = mean(mean(testYuv(:,:,2)));
null = mean(mean(nullYuv(:,:,2)));

normalisedTest = (testYuv(:,:,2)/test);
normalisedNull = (nullYuv(:,:,2)/null);

impulse = zeros(size(testYuv(:,:,2)));
impulse(1,1) =1;

moireSqri = sqri(luminance,normalisedTest,impulse,X,Y);
nullSqri = sqri(luminance,normalisedNull,impulse,X,Y);

out = moireSqri - nullSqri;
```

3.4 Method

3.4.1 Cutting the display samples

The two display samples, about 50mm x 50mm were cut from waste panels. The first step is to remove the polariser and the compensation layer from both sides of the panel which takes considerable force. A line was then scribed[6] on the glass of both sides of the panel around the glue line and the border of the panel was nibbled off using pliers. This leaves a sandwich of glass containing liquid crystal which can be separated leaving the colour filter side only. The samples are then scribed and cut from this.

Once the piece of glass is cut it was mounted on an adjustable block on the bottom of the stage. The angle that the pixels on the first piece of glass relative to the second piece can be adjusted by turning the screws to control their relative height. These were adjusted so that the interference appeared vertical with respect to the display on the video camera.[7]

3.4.2 Collecting the Data

The experiment consisted of varying the distance between the first display sample, and

[6] The glass cutting proved rather difficult, because glass is too thin to cut with a standard hardended steel glass cutter. A certain amount of pressure has to be applied to form the required grove in the glass, however display glass is much thinner than window glass and breaks before reaching this pressure. The final solution was to use a small carbide grinding wheel, held in ones hands and moved along a streight edge, to impart a scratch on glass and then use pliers or fingers to break along the line. This method would suggest the use of a small diamond scriber in the future to score the glass.

[7] A video camera seemed to provide the best moire resutls - initially I tried collecting the data using the Basler using a video lens, but I couldn't seem to get any contrast in the fringes even after taking the exponential of the values.

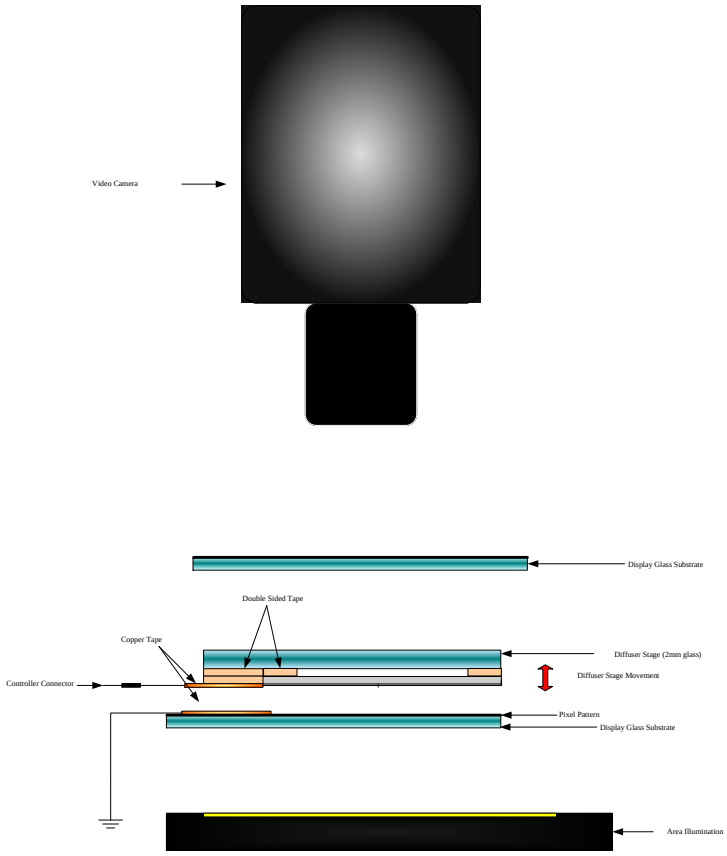


Figure 12: Experimental setup for Moire measurment. This time two vertical linear stages were used to control the distance between the displays and the distance of the diffuser from the rear display.

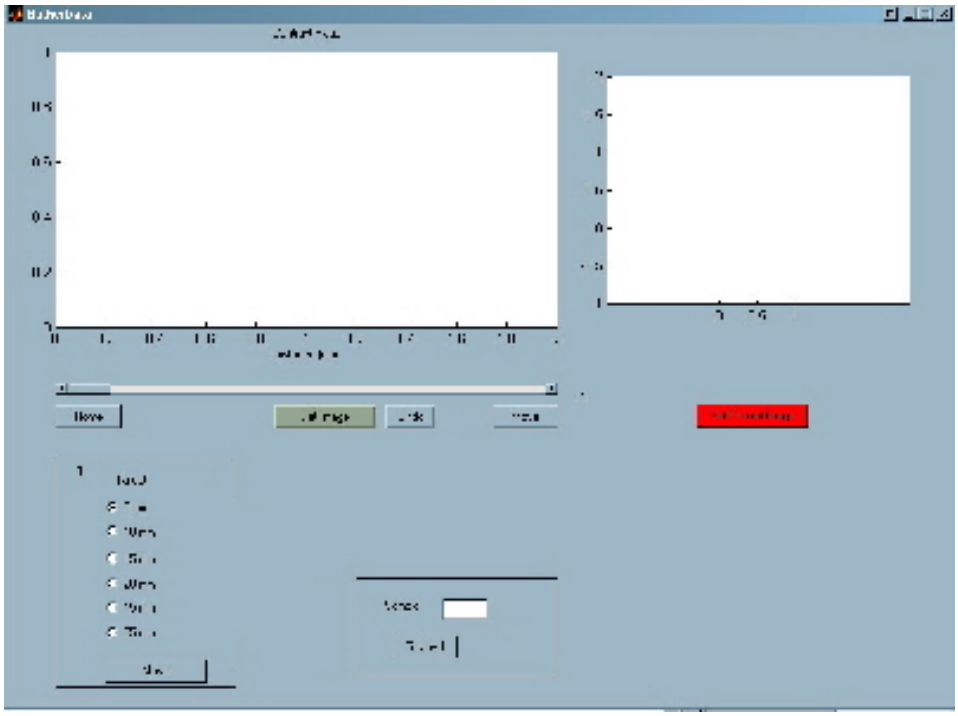


Figure 13: Moire Measurement GUI

each of 5 diffuser samples listed in Table: 2, from touching to 2mm increments of about 0.02mm; changing the distance between the display samples in increments of 5mm, and then varying the diffuser distance through its range again. The apparatus is shown in Figure: 12 The data was collected through a graphical user interface (GUI) programmed in MATLAB® that controlled the camera, the linear stages and reported the graphed calculated Square Root Integral(SQRI) Vs. the diffuser distance and a representation of the image back to the user in real time. The user could record a single shot and move both the second display sample and the stage. The results were gathered "manually", in the sense that four or five images were taken with the diffuser at each distance, then the next distance was chosen by moving the slider. The GUI handled storing the SQRI difference metric, time stamp, image, and current distance between the display samples in a structure. The distance between the display samples was picked using the radio buttons. The video camera was set to auto-focus and the exposure was fixed, with a visual check ensuring that it was focussing on the Moire interference.

Even though the problems with the positioning equipment had been ironed out in the previous experiment there were still some issues and mistakes made in the process. The sample glass breaks easily, which happened at least once and a new piece had to be cut.

Also keeping the bottom display sample in-line proved somewhat difficult since the spring provided rotation, however also allowed unwanted vertical and lateral movement. The first video camera used could not be kept on for an indefinite amount of time in video mode and failed during the experiment setup and a replacement had to be hired. On the first attempt to get the trial working a two pairs of wires and contacts were needed to provide home points for both the diffuser and the top display which proved impossible to get to work correctly. To simplify the situation, the front sample was moved differentially between positions, rather than returning to a home point between each run. A faint image of the two screws showed up in the final image which turned out to be a mixed blessing: one would expect a small effect on the final results, but on the other hand, because the distance between the two screws was known it was possible to determine the camera magnification, which was unobtainable otherwise.

3.5 Analysis and Results

Figure: 15 shows the predicted results for the modelled moire interference. Note that the graphs abruptly bottom out and in some cases start to rise back up at the end, probably due to the numerical limitations of the model, which unfortunately makes it difficult to find the minimum SQRI difference metric for any given display type. This could possibly be improved by using a finer points for the model, however this consumes a lot of computer memory and could not be calculated on a PC in the time given. . The Square Root integral gets smaller as the distance between the modelled display panels increases. The Square Root Integral Difference compared the model image with moire interference present to a flat image with uniform luminance.

?? shows the an image of the predicted and measured moire interference. There is good agreement in terms of the colours and their relative pixel positions.

Figure: 15 shows the metric value for the modelled moire interference. Note that the metric abruptly bottoms out and in some cases start to rise back up at the end. The maximum value of the metric increases as the distance between the displays increases.

?? shows the metric value for the measured moire interference. There is a lot of noise in the data, which continues to increase as the distance between the displays increases. The metric bottoms out at a value of around 5, and decreases both as the distance between the diffuser increases and as the distance between the displays increases.

3.6 Discussion

The match between the measured moire interference and the predicted moire interference is quite striking in terms of both the colour and the position of the fringes. The metric calculated using the predicted moire interference and spatial filter behaves as expected for small distances of the diffuser: it declines as the distance from the diffuser to the rear display increases and decreases as the distance between the displays increases. However there is some strange behavior for as the diffuser display distance starts increasing beyond 0.5 mm. The metric rapidly flattens out and for some distances between the displays even

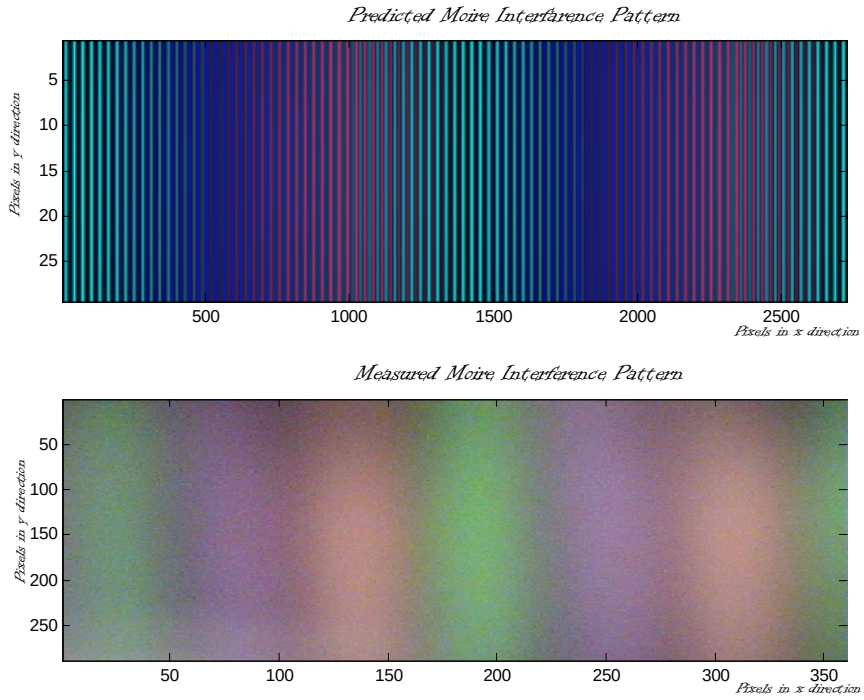


Figure 14: Predicted (top) and measured (bottom) Moire interference

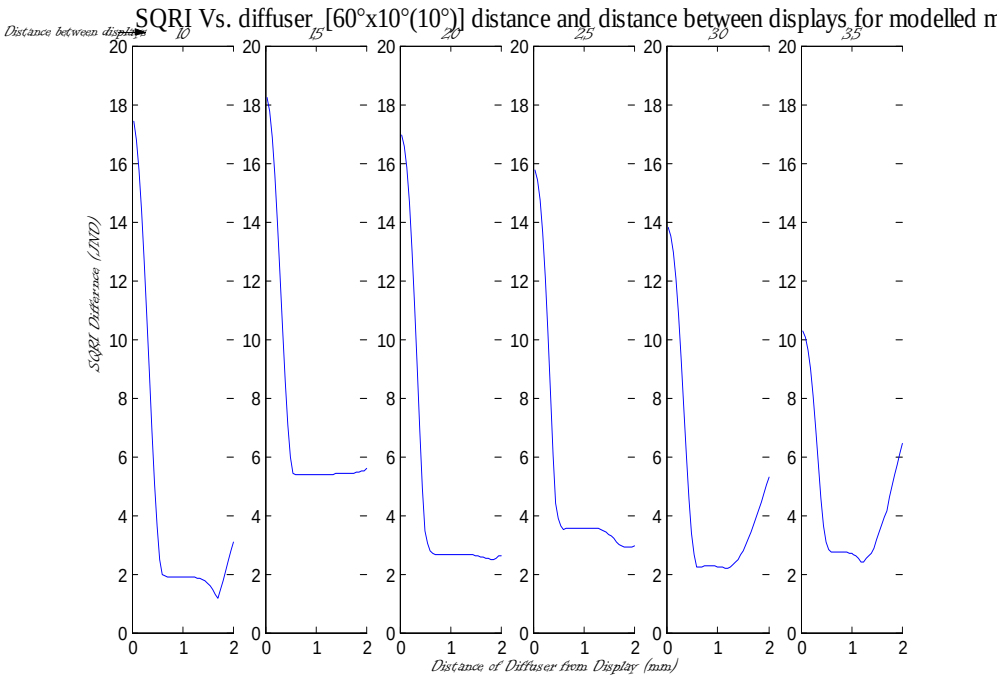


Figure 15: The predicted SQRI for the moire interference component of the image. Note the rapid flattening out of the graphs at bottom where the integral breaks down.

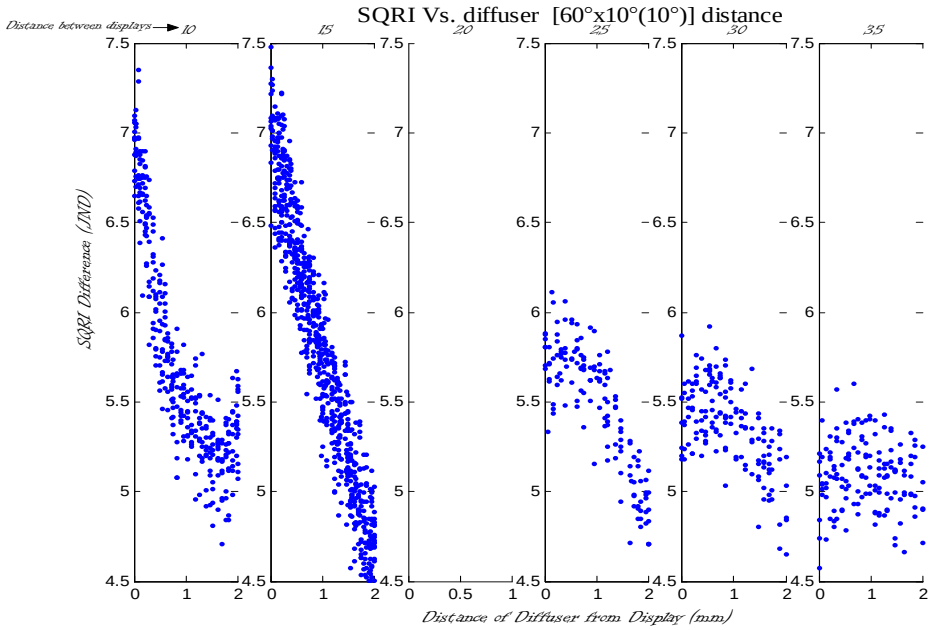


Figure 16: The Square Root Integral values of the Measured Images

begins to increase, which was not expected. Given that there is no pattern to this portion of the results one suspects the numerical model breaks down as the contrast begins to get very low, perhaps a finer grid would help to offset this. Because of this fact it is very difficult to tell what the distance is going to be for the zero crossing - if the true metric continues to fall a this rate then the zero crossing for most would most likely occur at about 1 mm. However if the true metric flattened out in a Gaussian like manner, as shown in the measured values, and which is likely given the nature of the spatial filtering, then it is likely that this zero crossing value could be as high as 2mm.

The metric calculated using the measured moire interference and the spatial filter is taken from very noisy data due to having to use a cheap camera, however there are some very clear features emerging here. In this case it is very difficult to measure the metric when the distance between the panels gets very large, from the graphs it is quite clear that the relationship between diffuser distance and the value of the metric is nonexistent when the panels are about 35mm apart, rather the points are just spread around about the bottom value. There is a strong case here for increasing the distance between the displays in the commercial monitor in the absence of other limiting factors. The metric value also bottoms out at about 5 JND's, which is probably due to the $\cos^4(x)$ fall off of the video lens. Given more time one could fit a polynomial surface to the image to account for this. The camera also has a terrible signal to noise ratio which is apparent when viewing mesh plots of the images, which would account for such noisy data. There is quite a difference in both the top and bottom values for the measured and expected values of the metric. This is probably due to not taking the luminance response of the camera into account fully, where the exponential of the mean luminance channel was used here to go some what towards addressing this issue. Of course one could calibrate the camera given sufficient time.

Chapter 4
Image Quality Survey

4.1 Objective

To determine the subjective image quality of a multi-layered display as a function of the distance to the diffuser to the rear image plane in terms of the moire interference, the quality of the rear image layer and the over-all impression of the over all image quality

4.2 Experimental Setup and Method

A prototype module Figure: 17 was constructed where the distance between the display layers, and the distance between the diffuser and the rear display could be adjusted. Potentially the distance between the two display layers which were two 15" flat panel displays, could be adjusted but it was decided to leave this set at 10mm. Unfortunately, because of time constraints a diffuser sheet out of a backlight was used, it would have been preferable to get a large sheet of holographic diffuser made up on a thin substrate. Because the front panel is only about 10% transmissive a high-brightened rear panel was used to provide enough light. Both sets of components that were to be moved were then mounted on drawer runners. Each of these subassemblies was mounted on the inside of a black enclosure. Linear stages were attached on the bottom of the enclosure and connected to the optical assemblies to provide the controlled movement. Micro switches were mounted on the inside of the enclosure, which, when contacted by the display layers, grounded a pin on the stage controllers triggering an external event which that the signaled that optical components were in the home position.

The apparatus was set-up in a shopping mall and the participants were given chocolate bars for their involvement. A GUI was also developed that provided a set of instructions, moved the stages, displayed an interface and underlying code that for collecting the responses. The participants were tested for colour blindness and 20/20 vision and were seated approximately 1-1.5 meters from the display. The GUI provided a small presentation on the two extremes of image quality by showing examples of a very blurry rear display Figure: 18 (bottom right) and Figure: 19 (3); with no moire present and a very clear rear display with a lot of moire present Figure: 18(bottom left) and Figure: 18 (2). A printer test photo shown in Figure: 20 below, contained a lot of fine detail and was placed as wall paper and used as a test image. Participants were instructed to note the fine detail in both this and the words and icons that were present on the desktop. The last section of the presentation consisted of three sliders where the participants supplied on a continuous scale their overall impression of image quality. The author and an assistant took turns at recruiting potential subjects and guiding them through the each of the trials.

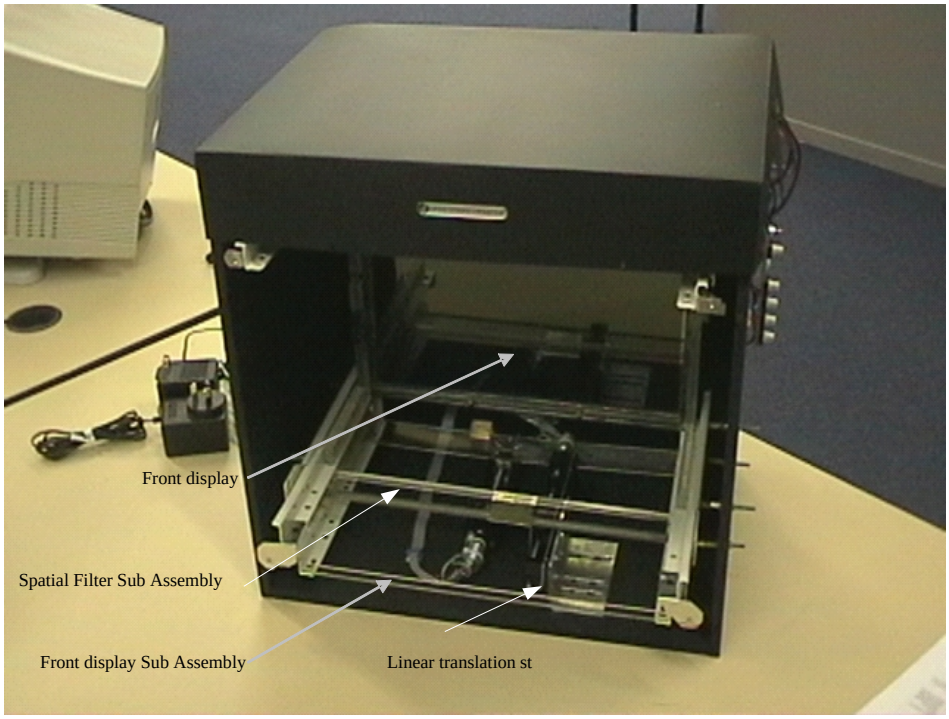


Figure 17: The apparatus designed and used in the image survey. Note in the actual survey there was a black insert that covered all the mechanical components.

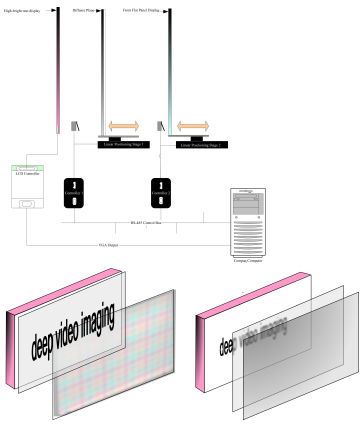


Figure 18: Schematic showing the setup of the experimental design

4.3 Results

The scatter plot in Figure: 21 shows the variance in image quality aspects, there is quite a wide distribution in the responses and from here one gets the sense that the image quality is mainly dependent on the clarity of the rear display.

The plot in Figure: 22 shows the averaged image quality aspects, where the error bars indicate the 95% confidence interval. It is clear here that the overall image quality follows the image clarity of the rear display.

4.4 Discussion

It is disappointing that the BTDF of the diffuser wasn't known means the models described in the previous sections cannot be referenced, and time has run out to measure the Moire interference and image clarity in the artefact which would be acceptable and included in a addendum to this report. However there is still useful information. The Moire interference is less salient to the general population than first expected and it appears the overall image quality is largely dependent on the clarity of the rear screen, so in terms of this trade-off one would lean towards making the image clearer at design time.

The variance in the results between subjects was not unexpected, and one wonders whether it may be better to use a threshold type experiment in the future to compare distances in the image quality which are the units in the square root integral difference metric. There are also some powerful statistical techniques to deal with this type of experiment, although it would take a lot of time to conduct since one needs a lot of subjects to define a level.

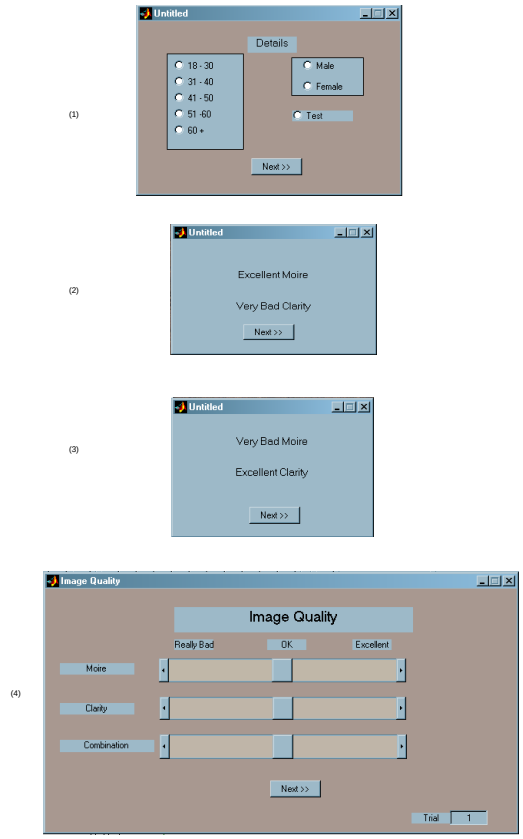


Figure 19: The graphcical user interface elements used in the survey.

Figure 20: Printer test photo dipslayed on screean and used for image quality survey.

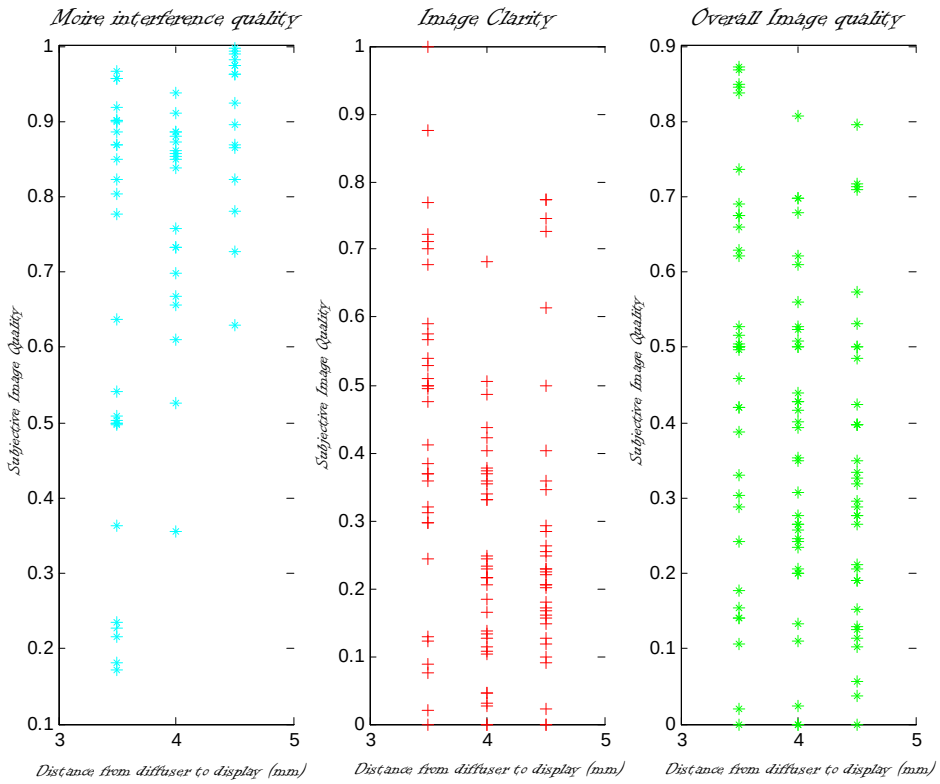


Figure 21: Scatter plot showing aspects of the subjective image quality

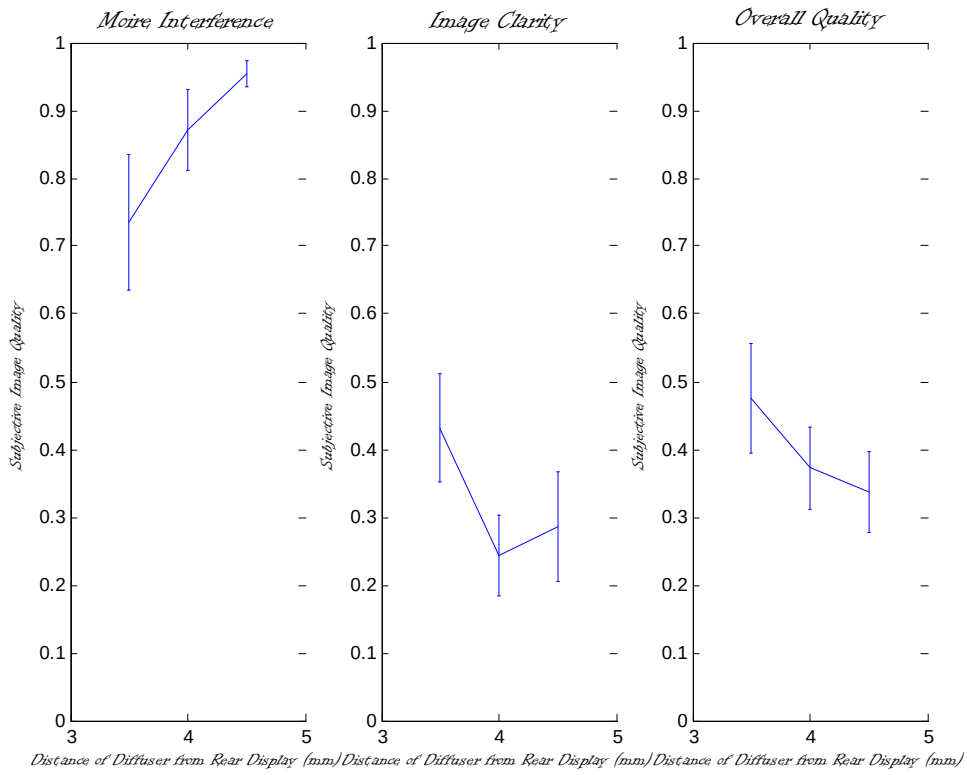


Figure 22: The average over the image quality sample. The error bars corrispond to the 95% confidence intervil.

Chapter 5
Conclusions

The Spatial filtering model is accurate to within an order of magnitude, in that it shows the correct behavior as the distance between the diffuser and the display increases and as the FWHM of the diffuser changes, however the result would benefit from some more careful experimental study. Time has run out Square Root Integral Metric with regard to the spatial filter, which is a very important component of this work. However this should not be difficult to calculate and will be completed in an addendum. The important information gained from this work is a sense of how tight the tolerance on the distance parameter is and this will be incorporated into the engineering knowledge base at Deep Video Imaging Ltd.

The Moire interference model works well in the sense that it predicts the colours and pitch of the interference fringes correctly. However this quantity is very difficult to measure well with a camera, it was almost impossible to detect on the expensive Basler which had a low signal to noise ratio in comparison to the consumer video camera that was eventually used. It is not understood as to why this should be the case, however one suspects this is because of the strong chromatic component to the fringes. In the future a better approach may be to use a calibrated, possible cooled, monochromatic camera to sample the fringes, where the subpixels would in the same direction on both displays which would produce monochromatic interference.

The Moire interference metric clearly needs more work. The measured results are very noisy, and logically the 1D Square Root integral should be used here instead of the 2D Square Root integral on the mean contrast ratio of the images, even though the former produced better results. There will be more work done over the next few months as part of the engineering effort, where a more practical approach to making these measurements will be investigated.

The importance of the clarity of the rear display image over the in the survey result was surprising and will be interesting to compare its contribution to the metric once the Square Root Integral of the spatial filter is measured.

Overall the project was ambitious in terms of the expected time frame, however given the importance of the measurements to a lot of approaches to 3D which still suffer the Moire interference problem, it would be worth the effort to polish the work further.

Appendix A
Spatial Filtering

5.1 Gatherdata Code

```
function varargout = GatherData(varargin)
    % GATHERDATA M-file for GatherData.fig
    % GATHERDATA, by itself, creates a new GATHERDATA or raises the existing
    % singleton*.
    % Last Modified by GUIDE v2.5 20-Sep-2003 18:03:26
    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name', mfilename, ...
     'gui_Singleton', gui_Singleton, ...
     'gui_OpeningFcn', @GatherData_OpeningFcn, ...
     'gui_OutputFcn', @GatherData_OutputFcn, ...
     'gui_LayoutFcn', [] , ...
     'gui_Callback', []);
    if nargin & isstr(varargin{1})
     gui_State.gui_Callback = str2func(varargin{1});
    end
    if nargout
     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
     gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT
    % — Executes just before GatherData is made visible.
    function GatherData_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
    % hObject handle to figure
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    % varargin command line arguments to GatherData (see VARARGIN)
    % Choose default command line output for GatherData
    handles.output = hObject;
    %==========================CONFIGURATION==========================
    %External Functions used:
    %movestage
    %————————————————————————————————-
    % Output Structure - handles.output.[]
```

Section A.1 Gatherdata Code

```
% e.g. - handles.ouput.Date
%-----------------------------------------------
% .date (time stamp)
% .image (image matrix)
% .target (Target Type): element of [2 4 6 8 10 12]
% .contrastRatio (Contrast ratio value): 0 < CR < 1
% .distance (Distance from stage to home in mm)
handles.contrast = cell(8,1);
handles.output = struct([]);
%-----------------------------------------------
% Temporary internal variables - handles.int.[]
%-----------------------------------------------
 handles.int.s1 =0; %(Distance from home of stage 1)
 handles.int.s2 =0; %(Distance from home of stage 2)
 handles.int.serialID = cportopen('com2');
 % .currentTarget %(The current target)
 handles.int.index = 1; %(Current index into the strucuture, iterates when an image is taken)
 set(handles.radiobutton1,'Value',1);
 handles.int.button = 1;

 X = 7.5/84*[ 0 27 41.5 55.5 62.8 69.5 76.5 84 ];
handles.int.TARGET = X(1:(length(X)-1));
%-----------------------------------------------
%Notes
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes GatherData wait for user response (see UIRESUME)
uiwait(handles.figure1);
% — Outputs from this function are returned to the command line.
function varargout = GatherData_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
%-----------------------------------------------
%GET IMAGE CODE-------------------------------
%-----------------------------------------------
% — Executes on button press in GetImage.
function GetImage_Callback(hObject, eventdata, handles)
%Set image limits
IMAGE_X_MIN = 70;
IMAGE_X_MAX = 480;
IMAGE_Y_MIN = 1;
```

Appendix A Spatial Filtering

```
%Select the viewing window
switch handles.int.button
 case {1}
 IMAGE_Y_MAX = 640;
 case {2,3}
 IMAGE_Y_MAX = 620;
 case {4,5,6}
 IMAGE_Y_MAX = 300;
end
%Get and store the picture
tempIndex = handles.int.index;
x = zeros(1,5);
for i = 1:5
 tempImage = get_yuv;
 x(i) = contrastRatio(tempImage);
end
CR = mean(x)
handles.output(tempIndex).image = (tempImage);
handles.output(tempIndex).date = clock;
handles.output(tempIndex).currentTarget = handles.int.button;
handles.output(tempIndex).keep = 1;
% Calculate and store the contrast ratio
tempImage = handles.output(tempIndex).image(IMAGE_Y_MIN:IMAGE_Y_MAX,IMAGE_X_MIN:IMAGE_X_MAX);
handles.contrast{handles.int.button,1} = [handles.contrast{handles.int.button,1};[get(handles.Slider,'Value'),CR]]
handles.output(tempIndex).CR = [get(handles.Slider,'Value'),CR];
hold on
axes([handles.axes1]) % Select the proper axes
cla
%Plot the contrast ratio
set(handles.axes1,'XLim',[0 2]);
set(handles.axes1,'Units','centimeters')
set(handles.axes1,'Position',[1.295 9.994 13.246 7.244])
set(handles.axes1,'NextPlot','add')
markers = '.ox+*s';
for i = 1:8
 if ~isempty(handles.contrast{i,1})
 h = line(handles.contrast{i,1}(:,1),handles.contrast{i,1}(:,2));
 set(h,'Marker',markers(i))
 end
end
handles.int.currentLine = h;
%Write the image
axes([handles.axes2])
cla
imagesc(tempImage')
```

Section A.1 Gatherdata Code

```
%Update the index
handles.int.index = handles.int.index + 1;
% Update handles structure
guidata(hObject, handles);
%————————————————————
% — Executes on button press in Undo.
function Undo_Callback(hObject, eventdata, handles)
%Needs fleshing out after doing movement code
tempIndex = handles.int.index - 1
%Set the keep value to zero
handles.output(tempIndex).keep = 0;
%Remove the previous point and replot
handles
[len,x] = size(handles.contrast{handles.int.button,1})
len = len -1;
if len==0
 handles.contrast{handles.int.button,1} =[]
 axes([handles.axes2])
 cla
end
if len~=0
 handles.contrast{handles.int.button,1} = [handles.contrast{handles.int.button,1}(1:len,:)]
end
markers = ['.ox+*s'];
axes([handles.axes1]); % Select the proper axes
cla;
for i = 1:8
 if ~isempty(handles.contrast{i,1})
 h = line(handles.contrast{i,1}(:,1),handles.contrast{i,1}(:,2));
 set(h,'Marker',markers(i));
 end
end
handles.int.index = tempIndex;
guidata(hObject, handles);
%————————————————————————————————————-
%MOVE 1 CODE————————————————————————————
%OK
%————————————————————————————————————-
% — Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
handles.int.button = 1;
off = [handles.radiobutton2,handles.radiobutton3,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
```

```
guidata(hObject, handles);
% — Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
handles.int.button = 2;
off = [handles.radiobutton1,handles.radiobutton3,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
handles.int.button = 3;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
handles.int.button = 4;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
handles.int.button = 5;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton4,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
handles.int.button = 6;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton4,handles.radiobutton5];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
%————————————————————————————————————
% — Executes on button press in Home1.
function Home1_Callback(hObject, eventdata, handles)
set(handles.Moving1,'String','Moving.....')
```

Section A.1 Gatherdata Code

```
stagehome(handles.int.serialID,1,1)
set(handles.Moving1,'String',' ')
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton4,handles.radiobutton5,handles.radiobutton6];
handles.int.button = 1;
mutual_exclude(off)
set(handles.radiobutton1,'Value',1);
handles.int.s1 = 0;
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in Move1.————————————-
function Move1_Callback(hObject, eventdata, handles)
targetValue = handles.int.button;
moveStageDelta(handles.int.s1,handles.int.TARGET(targetValue),1,handles)
handles.int.s1 = handles.int.TARGET(targetValue);
guidata(hObject, handles);
%————————————————————————————————————-
%MOVE 2 CODE————————————————————————————
%In Testing
%————————————————————————————————————-
function Slider_Callback(hObject, eventdata, handles)
% — Executes on button press in Move2.
function Move2_Callback(hObject, eventdata, handles)
moveStageDelta(handles.int.s2,get(handles.Slider,'Value'),2,handles)
handles.int.s2 = get(handles.Slider,'Value');
% Update handles structure
guidata(hObject, handles);
% — Executes during object creation, after setting all properties.
function Slider_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background, change
% 'usewhitebg' to 0 to use default. See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
 set(hObject,'BackgroundColor',[.9 .9 .9]);
else
 set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% — Executes on button press in Home2.
function Home2_Callback(hObject, eventdata, handles)
%Move the stage home and let the user know it is moving
set(handles.Moving2,'String','Moving.....')
stagehome(handles.int.serialID,2,-1)
set(handles.Moving2,'String',' ')
%Set the slider to the home position
set(handles.Slider,'Value',0);
```

```
%Update the tracking value
handles.int.s2 = 0;
% Update handles structure
guidata(hObject, handles);
%ADMINISTRATION CODE————————————————————
% — Executes during object creation, after setting all properties.
function SampleName_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
 set(hObject,'BackgroundColor','white');
else
 set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function SampleName_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'String') returns contents of SampleName as text
% str2double(get(hObject,'String')) returns contents of SampleName as a double
% — Executes on button press in Finished.
function Finished_Callback(hObject, eventdata, handles)
cportclose(handles.int.serialID)
guidata(hObject, handles);
% UIWAIT makes GatherData wait for user response (see UIRESUME)
uiresume(handles.figure1);
%————————————————————————————————————-
%SUB FUNCTIONS
%————————————————————————————————————-
%————————————————————————————————————-
function moveStageDelta(presentPosition, finalPosition, button, handles)
switch button
 case 1
 %Make sure the push-button stays down
 set(handles.Move1,'Value',0);
 %Tell the user the stage is moving
 set(handles.Moving1,'String','Moving.....');
 %Move the stage
 moveStage(handles.int.serialID,1,-(finalPosition-presentPosition));
 %Delete the 'Moving' text
 set(handles.Moving1,'String',' ');
 % Bring the button back up
 set(handles.Move1,'Value',1);

 case 2
 %Make sure the push-button stays down
 set(handles.Move2,'Value',0);
 %Tell the user the stage is moving
```

```
 set(handles.Moving2,'String','Moving.....');
 %Move the stage
 moveStage(handles.int.serialID,2,(finalPosition-presentPosition));
 %Delete the 'Moving' text
 set(handles.Moving2,'String',' ');
 % Bring the button back up
 set(handles.Move2,'Value',1);
end

%--------------------------------------------------------
function mutual_exclude(off)
set(off,'Value',0);
%--------------------------------------------------------
%Scan Data
% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
nSamples = 50
for i = 1 : nSamples + 1

 %Set image limits
 IMAGE_X_MIN = 70;
 IMAGE_X_MAX = 480;
 IMAGE_Y_MIN = 1;

 %Select the viewing window

 switch handles.int.button
 case {1}
 IMAGE_Y_MAX = 640;
 case {2,3}
 IMAGE_Y_MAX = 620;
 case {4,5,6}
 IMAGE_Y_MAX = 300;
 end

 %Get and store the picture
 tempIndex = handles.int.index;

 x = zeros(1,5);
 for i = 1:5
 tempImage = get_yuv;
 x(i) = contrastRatio(tempImage);
```

```
end

CR = mean(x)

handles.output(tempIndex).image = (tempImage);
handles.output(tempIndex).date = clock;
handles.output(tempIndex).currentTarget = handles.int.button;
handles.output(tempIndex).keep = 1;

% Calculate and store the contrast ratio
tempImage = handles.output(tempIndex).image(IMAGE_Y_MIN:IMAGE_Y_MAX,IMAGE_X_MIN:IMAGE_X_MAX);
handles.contrast{handles.int.button,1} = [handles.contrast{handles.int.button,1};[get(handles.Slider,'Value'),CR]]
handles.output(tempIndex).CR = [get(handles.Slider,'Value'),CR];

hold on
axes([handles.axes1]) % Select the proper axes
cla

%Plot the contrast ratio
set(handles.axes1,'XLim',[0 2]);
set(handles.axes1,'Units','centimeters')
set(handles.axes1,'Position',[1.295 9.994 13.246 7.244])
set(handles.axes1,'NextPlot','add')

markers = '.ox+*s';

for i = 1:8
if ~isempty(handles.contrast{i,1})
h = line(handles.contrast{i,1}(:,1),handles.contrast{i,1}(:,2));
set(h,'Marker',markers(i))
end
end

handles.int.currentLine = h;

%Write the image
axes([handles.axes2])
cla
imagesc(tempImage')

%Update the index
handles.int.index = handles.int.index + 1;
%Move the stage
movestage(handles.int.serialID,2,(2/nSamples))
```

```
 %Update the slider
 x = get(handles.Slider,'Value');
 set(handles.Slider,'Value',(x + (2/nSamples)))
end
% Update handles structure
guidata(hObject, handles);
beep
```

5.2 Movestage Code

```
function moveStage(serial_id,node,x)
    %————————moveStage(serial_id,node,x)————————————%
    %Moves stage x millimeters
    %————————————————-Variables————————————————-%
    %serial_id - Structure generated by serial function in MATLAB
    %node - The number of the controller being used 1,2,3,4 etc
    %x - Distance in millimeters that the stage needs to be moved. +ve is
    % towards motor end of controller
    %————————————————————————————————————————%
    % Gareth Bell <gareth.bell@deepvideo.com> %
    % Date Created 13/05/03 %
    % Date Last Edited 9/06/03 %
    % Revision 2.0 %
    %————————————————Notes————————————————-%
    %Extensively reworked after using velocity rather than position mode
    %_____________________________________________________________________________%
    F = 5.644e-3; %Fudge factor
    if x~=0
     v = sign(x)*30 %Velocity
     t = abs(x)/(abs(v)*F) %Calculated time
     cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,v),'CR')
     pause(t)
     cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,0),'CR')
    else
     cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,0),'CR')
    end
```

5.3 Stagehome Code

```
%————————moveStage(serial_id,node,direction)————————————%
    %Moves stage to home hardstop
    %————————————————————-Variables————————————————-%
    %serial_id - Structure generated by serial function in MATLAB
```

Appendix A Spatial Filtering

```
%node - The number of the controller being used 1,2,3,4 etc
% - -1 or 1
% Positive is towards motor end of controller, can be negetive
%
%————————————————————————————————————%
% Gareth Bell <gareth.bell@deepvideo.com> %
% Date Created 5/06/03 %
% Date Last Edited 7/06/03 %
% Revision 1.0 %
%————————————————Notes————————————————-%
%____________________________________________________________________________%

function stageHome(serial_id,node,direction)


%check for bad input
if abs(direction)>1
direction=sign(direction)
warning('direction should be 1 or -1')
end

%Check to see if home already
status = 0;
cportreset(serial_id)

cportwrite(serial_id,sprintf('%1.0f ST',node),'CR');
pause(0.1)
status = cportgetchar(serial_id,13)
if (bitand(hex2dec(status(7:11)),4096))
error('!!!!!!!!!Already Home!!!!!!!!!!!')
end

%Seek home position
cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,(direction*20)),'CR')
status = 0;
cportreset(serial_id)
while (sum(size(status))~=14)|~(bitand(hex2dec(status(7:11)),4096))
cportwrite(serial_id,sprintf('%1.0f ST',node),'CR');
pause(0.1)
status = cportgetchar(serial_id,13);
fprintf('|')
end

%Back off for a second
cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,(direction*-20)),'CR')
```

```
pause(1)
cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,0),'CR')

%Re-seek but this time slower for more accuracy
pause(0.5)
cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,(direction*2)),'CR')
status = 0;
cportreset(serial_id)
while (sum(size(status))~=14)|~(bitand(hex2dec(status(7:11)),4096))
cportwrite(serial_id,sprintf('%1.0f ST',node),'CR');
status = cportgetchar(serial_id,13);
pause(0.1)
fprintf('+')
end
cportwrite(serial_id,sprintf('%1.0f V %1.0f',node,0),'CR')

%Define this point as home
cportwrite(serial_id,'1 HOME','CR')
fprintf('*')
```

5.4 get_yuv Code

```
%[y,u,v] = get_yuv()
    %
    % Capture an image from the camera, and return Y,U and V color
    % components as seperate 640x480 matrices
    function [y,u,v] = get_yuv()
     a=cam;
     arraysize = size(a);
     rows = arraysize(1);
     y=zeros(640,480);
     u=zeros(640,480);
     v=zeros(640,480);
     for i=1:(rows/2)
     y(i,:) = a(2*i,:);
     u(i,:) = bitand(a(2*i-1,:), hex2dec('F0'))-128;
     v(i,:) = bitand(a(2*i-1,:), hex2dec('0F'));
     end
    end
```

5.5 btdfFilter Code

```
function [PSF] = btdfFilter(areaLens,OD,OL,sizeFilter,M,cameraPixelSize,gridAlpha,gridThetaIn,BTDF,p)
```

```
%function [PSF] = btdfFilter(areaLens,OD,OL,sizeFilter,t,n,M,cameraPixelSize,gridAlpha,gridThetaIn,BTDF,plot)%
%————————————————-Variables————————————————-%
%————————————————————————————————————%
% Gareth Bell %
% Date Created 27/09/03 %
% Date Last Edited 27/09/03 %
% Revision 1.0 %
%————————————————Notes————————————————-%
%____________________________________________________________________________%
%STAY IN OBJECT DOMAIN
%Calculate the distance between the diffuser and the lens
DL = OL - OD;
%Divisions along the object domain
deltaX = -(cameraPixelSize/M)*sizeFilter:(cameraPixelSize/M):(cameraPixelSize/M)*sizeFilter;
%Calculate the angles
thetaInT = atan(deltaX./OD)*180/pi;
thetaOutT = atan(deltaX./DL)*180/pi;
alpha = thetaInT + thetaOutT;
thetaIn = -abs(thetaInT);
%Get the filer and plot the graph
fs = griddata(gridAlpha,gridThetaIn,BTDF,alpha,thetaIn,'cubic').*cos(thetaIn*pi/180);
PSF =fs*areaLens^2./((deltaX.^2 +OD^2).*(deltaX.^2 + DL^2))/M^2;
%Plot the graph
if p
 h = gcf;
 hold on
 figure(h);
 subplot(1,3,2),plot3(alpha,thetaIn,fs,'.')
 xlabel('thetaIn (degrees)')
 ylabel('alpha (degrees)')
 title('Angle In Vs Angle out of diffuser')

 h = gcf;
 figure(h);
 subplot(1,3,3),plot(deltaX*5,PSF);
 title('Calculated Image Filter')
 xlabel('Distance across CCD (m)')
 ylabel('Spread function')
end
```

5.6 Calculatefilter Code

```
function [PSF] = calculateFilter(distance,diffuser,orientation,p)
    warning off MATLAB:polyfit:RepeatedPointsOrRescale
```

Section A.6 Calculatefilter Code

```
warning off MATLAB:divideByZero
%Put in the perameters
areaLens = 7.1732e-005;
OD = distance;
OL = 34e-3;
t = 3e-3;
n = 1.5;
M = 5;
cameraPixelSize = 8.3e-6;
sizeFilter = 320;
hold on
%Load data
[angle,F] = processing(diffuser,orientation);
thetaIn = -[0, 15, 30, 45];
domain = min(angle)-1*15:max(angle)+4*15;
coeffs = zeros(3,4);
%Find when log(F) is at half the maximum of log(F)
positions = F > ones(length(F),1)*max(F)*0.5;
%Fit a curve to each of the graphs
for i = 1:4
 [tempCoeffs,s] = polyfit([angle(positions(:,i))-thetaIn(i)], [log(F(positions(:,i),i))],2);
 coeffs(:,i) = tempCoeffs';
 alpha = [min(angle):max(angle)]-(thetaIn(i));
 [fit(:,i)] = [zeros(15*i,1);exp(polyval(coeffs(:,i),alpha))';zeros(15*(5-i),1)];
end
if p
 %Plot the fitted curve and the data if required
 h = gcf;
 figure(h)
 subplot(1,3,1),plot(domain,fit);
 h = gcf;
 figure(h)
 hold on;
 subplot(1,3,1), plot(angle-thetaIn(1),F(:,1),'x',angle-thetaIn(:,2),F(:,2),'x',angle-thetaIn(:,3),F(:,3),'x',angle-
thetaIn(:,4),F(:,4),'x');
 xlabel('alpha (degrees)');
 ylabel('BDTF');
 title('Fitted BDTF against alpha');
 hold off;
end
%Form the interpolated function between the curves and graph it if required
[gridAlpha,gridThetaIn] = meshgrid(domain,thetaIn);
BTDF = fit';
[XI,YI] = meshgrid(domain,[0:-1:-45]);
ZI = griddata(gridAlpha,gridThetaIn,BTDF,XI,YI,'cubic');
```

```
%Plot the graph
if p
 h = gcf;
 figure(h);
 subplot(1,3,2), surf(domain,[0:-1:-45],ZI);
 xlabel('alpha (degrees)')
 ylabel('thetaIn (degrees)')
 zlabel('BTDF')
 title('Interpolated BTDF function along 10 degree orientaion for 60x10 holographic diffuser')
end
%Calculate the filter
[PSF] = btdfFilter(areaLens,OD,OL,sizeFilter,M,cameraPixelSize,gridAlpha,gridThetaIn,BTDF,p);
```

5.7 compareresults code

```
function [positions,measuredCR,calculatedCR] = compareResults(x,diffuser,orientation,nullFileName);
    %Predict the calculated contrast ratio
    %Intialise Matricies
    calculatedCR = zeros(50,6)*nan;
    distance = [];
    modelTarget = createModelTarget(nullFileName,0);
    for i = 5:50
     distance(i) = x(i).CR(1)/1000;
     filter = calculateFilter(distance(i),diffuser,orientation,0);
     index = ~isnan(filter);
     filter = filter(index);
     filteredTarget = imfilter(modelTarget,filter');
     filteredTarget = filteredTarget(500:1140,:);
     for j = 1:6
     switch j
     case {1}
     IMAGE_Y_MAX = 640;
     case {2,3}
     IMAGE_Y_MAX = 620;
     case {4,5,6}
     IMAGE_Y_MAX = 300;
     end
     calculatedCR(i,j) = contrastRatio(filteredTarget(1:IMAGE_Y_MAX,j));
     end
    end
    %Plot the predicted contrast ratio
    figure
    subplot(1,2,1),plot(repmat([distance]',1,6),calculatedCR);
    title('Predicted Contrast Ratio')
```

```
xlabel('Target - diffuser distance (mm)')
ylabel('Contrast Ratio')
hold on
%Plot the measured contrast ratio
measuredCR = [x.CR];
positions = [find(measuredCR==0),length(measuredCR)];
for i = 1:length(positions)-1;
 left = positions(i)+1;
 right = positions(i+1)-1;
 subplot(1,2,2),plot(([measuredCR((left-1):2:(right-1))]),measuredCR(left:2:right),'x');
 hold on;
end
title('Measured Contrast Ratio')
xlabel('Target - diffuser distance (mm)')
ylabel('Contrast Ratio')
markers = '.ox+*s'
%plot the measured contrast ratio against predicted contrast ratio
figure
if length(positions)<=7
 for i = 1:length(positions)-1;
 left = positions(i)+1;
 right = positions(i+1)-1;
 plot(([calculatedCR(1:length(measuredCR(left+2:2:right)),i)]-2.5e-3),measuredCR(left+2:2:right),markers(i));
 hold on;
 end
else
 for i = 1:6;
 left = positions(i)+1;
 right = positions(i+1)-1;
 plot(([calculatedCR(1:length(measuredCR(left+2:2:right)),i)]-2.5e-3),measuredCR(left+2:2:right),markers(i));
 hold on;
 end
end
title('Measured Values Vs. Expected Values')
xlabel('Calculated Contrast Ratio')
ylabel('Measured Contrast Ratio')
```

5.8 convertData code

```
function convertData(number,orientation)
    colour = ['GRB'];
    inputAngle = ['00','15','30','45'];
    if orientation ==1
     orintationSymbol = 'H'
```

```
else
 orintationSymbol = 'V'
end

for i = 1:4
 for j = 1:3
 baseName = strcat('DVI_',num2str(number), orintationSymbol,'_',colour(j),'_',inputAngle(2*i-
1),inputAngle(2*i));
 inputName = strcat(baseName,'.brdf')
 outputName = strcat(baseName,'.mat');
 data = dlmread(inputName,'\t',1,0);
 save(outputName,'data')
 end
end
```

5.9 calculateModelTarget Code

```
function modelTarget = createModelTarget(nullFileName,p)
    %————————moveStage(InputImage)————————————%
    %Calculates a 1D model target given images of the real target
    %————————————————-Variables————————————————-%
    % Matrix of Luminaninance Values
    %————————————————————————————————————%
    % Gareth Bell <gareth.bell@deepvideo.com> %
    % Date Created 01/08/03 %
    % Date Last Edited 01/08/03 %
    % Revision 1.0 %
    %————————————————Notes————————————————-%
    load(nullFileName);
    SIZE_PIXEL = 8.3e-6; %The size of the pixel in the Basler
    mm_PER_m = 1000; %converstion factor between mm and m
    M = 5; %The magnification of the camera lens
    SPATIAL_FREQUENCY = [2,4,6,8,10,12] * mm_PER_m * SIZE_PIXEL /M; %This gives spatial
frequency on the CCD in cycles per pixel
    k = 2*pi*SPATIAL_FREQUENCY; %convert this into a save number
    meanImage = zeros(5*640,6); %Initialise the meanImage matrix
    targetMin = zeros(1,6); %Intialise the vector that stores the miniumum value of the target image
    targetMax = zeros(1,6); %Intialise the vector that stores the maxiumum value of the target image
    newDomain = [1:5*640]';
    Tmax = 1; %The maximum of our generated target
    modelTarget = zeros(length(newDomain),6);
    for i = 1:6

     switch i
```

Section A.9 calculateModelTarget Code

```
case {1}
IMAGE_Y_MAX = 640;
case {2,3}
IMAGE_Y_MAX = 620;
case {4,5,6}
IMAGE_Y_MAX = 300;
end
meanImage = mean([x(i).image(1:IMAGE_Y_MAX,:)']); %Find the mean value of the columnns and
store in a vector
[targetMin(i),targetMax(i)] = findExtreme(meanImage); %Find the maximum and minimum value of the
target
domain = 1:length(meanImage);

%Plot the target and the limits if desired
if p
y = ones(1,length(domain));
subplot(1,6,i)
plot(meanImage);
hold on;
subplot(1,6,i),plot(targetMin(i)*y,'red');
subplot(1,6,i),plot(targetMax(i)*y,'blue');
title(strcat('Mean image of target_',num2str(i)));
xlabel('distance (pixels)');
ylabel('Normalised response');
end
A = targetMax(i) - targetMin(i);
modelTarget(:,i) = [0.5*(sin(k(i)*newDomain)+1)*(A/targetMax(i)) + targetMin(i)/targetMax(i)]
;%calculate the model target values and store in a matrix
%which is nice coz you can just filter the matrix
end
```

Appendix B
Moire Interference

5.1 MoireSQRITheory Code

```
function out = moireSqri(luminance,testImage,nullImage,X,Y)
    %_____________________________________________________________________%
    %———————out = moire(testImage,nullImage,X,Y)————————%
    %calculates the square root integral of the moire interfarence present
    %————————————————-Variables———————————-%
    % testImage - rgb image with moire interfarence present
    % nullImage - image using same camera on flat field of same luminance
    % X - size of the image in the x direction in millimeters
    % Y - size of the image in the y direction in millimeters
    %
    %———————————————————————————%
    % Gareth Bell %
    % Date Created 13/09/03 %
    % Date Last Edited 13/09/03 %
    % Revision 1.0 %
    %————————————————Notes————————————————-%
    % Calculates the SQRI in the vertical direction only. Nice thing is that
    % you can measure the average luminance using a light meter and everything
    % else is spatial frequency dependent
    %_____________________________________________________________________%
    testYuv = mean(testImage);
    nullYuv = mean(nullImage);
    test = mean(mean(testYuv(:,:,2)));
    null = mean(mean(nullYuv(:,:,2)));
    normalisedTest = (testYuv(:,:,2)/test);
    normalisedNull = (nullYuv(:,:,2)/null);
    impulse = zeros(size(testYuv(:,:,2)));
    impulse(1,1) =1;
    moireSqri = sqri(luminance,normalisedTest,impulse,X,Y);
    nullSqri = sqri(luminance,normalisedNull,impulse,X,Y);
    out = moireSqri - nullSqri;
```

5.2 SQRI Code

```
function I = SQRI(Luminance,testImage,referenceImage,X,Y)
```

Section B.2 SQRI Code

```
%————SQRI(Luminance,testImage,referenceImage_0,X,Y)————-%
%gareth.bell@deepvideo.com
%
%Calculate the square root integral metric for a viewing distance of
%570mm given a reference and a distorted image.
%————————————-Variables————————————-%
% Luminance - Average luminance of the images (this should be the same
%for both)
% testImage - The image to be compared
% referenceImage - The reference image
% X - The size of the image in the x direction in mm
% Y - The size of the image in the y direction in mm
%————————————————————————————————————%
% Gareth Bell %
% Date Created 27/08/03 %
% Date Last Edited 28/08/03 %
% Revision 1.0 %
%————————————Notes————————————-%
%See Quality Aspects of Computer based Video Services for details
%Major revision and simplification given Peter Barten's presentation pg 27
%____________________________________________________________________%
VIEWING_DISTANCE = 570; %Standard viewing distance
%see http://www.cquest.utoronto.ca/psych/psy280f/ch5/sf.html
% Find the size of the image
[nPixels, mPixels] = size(referenceImage);
%Convert the input size into degrees assuming viewing distance of 570mm
thetaX = atan(X / VIEWING_DISTANCE) * 180/ pi;
thetaY = atan(Y / VIEWING_DISTANCE) * 180/ pi;
%Calculate the size of the increments
deltaU = 1 / thetaX;
deltaV = 1 / thetaY;
%Do the fast fourior transforms of both images
F_testImage = fft2(testImage);
F_referenceImage = fft2(referenceImage);
%Find the modulation transfer function
mtf = abs(F_testImage ./ F_referenceImage);
%We need to create a grid of u and v values
[U, V] = meshgrid( [deltaU:deltaU:(mPixels * deltaU)] , [deltaV:deltaV:(nPixels * deltaV)] );
%Calculate the frequency into the 1D Contrast sensitivity function
frequency = sqrt( U .^2 + V .^2 );
%Calculate the contrast sensitivity function
localCsf = csf(frequency, Luminance, thetaX, thetaY);
%This needs to be integrated over a log grid
logGrid = 1 ./ ( U .^2 + V .^2 );
%Calculate the final integral
```

```
I = 1./(2 * pi * log(2) )...
 * sum( sum( sqrt(mtf .* localCsf) .* logGrid * deltaU * deltaV ) );
```

5.3 Contrast Ratio

```
function CR = ContrastRatio(inputImage)
%————————moveStage(InputImage)————————————————%
%Calculates contrast ration of a given image
%————————————————-Variables————————————————-%
% Matrix of Luminaninance Values
%————————————————————————————————————%
% Gareth Bell <gareth.bell@deepvideo.com> %
% Date Created 11/08/03 %
% Date Last Edited 11/08/03 %
% Revision 1.0 %
%————————————————Notes————————————————-%
%Get the mean of the image
meanImage = mean(inputImage,2);
%Calculate the contrast ratio
maxPos = lclmax(meanImage,3)
minPos = lclmax(-meanImage,3)
max = mean(meanImage(maxPos))
min = mean(meanImage(minPos))
CR= max/min
%————————————————————————————————————
function pk = lclmax(x,p2)
%LCLMAX [ A.K.Booer 3-Dec-1992 ]
%
% pk = lclmax(x,p) 1-D peak search down columns
% using peak half window width of p.
%
% Returns PK, a BOOLEAN matrix showing local maxima positions.
%
p = 2*p2 + 1; % full window width
[n,m] = size(x); % useful dimensions
z = zeros(p2,m); % pre-allocate result
i = toeplitz(p:-1:1,p:n); % shift operator
%
y = zeros(p,(n-p+1)*m); % temporary matrix
y(:) = x(i,:); % index into original data matrix
ma = max(y); % find maximum in window
%
pk = [z ; reshape(ma,n-p+1,m) ; z]; % add missing edge elements
pk = pk == x; % find matching elements
```

```
%%
```

5.4 gatherData

```
function varargout = GatherData(varargin)
    % GATHERDATA M-file for GatherData.fig
    % GATHERDATA, by itself, creates a new GATHERDATA or raises the existing
    % singleton*.
    % Last Modified by GUIDE v2.5 14-Sep-2003 18:00:46
    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name', mfilename, ...
     'gui_Singleton', gui_Singleton, ...
     'gui_OpeningFcn', @GatherData_OpeningFcn, ...
     'gui_OutputFcn', @GatherData_OutputFcn, ...
     'gui_LayoutFcn', [] , ...
     'gui_Callback', []);
    if nargin & isstr(varargin{1})
     gui_State.gui_Callback = str2func(varargin{1});
    end
    if nargout
     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
     gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT
    % — Executes just before GatherData is made visible.
    function GatherData_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
    % hObject handle to figure
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    % varargin command line arguments to GatherData (see VARARGIN)
    % Choose default command line output for GatherData
    handles.output = hObject;
    %==========================CONFIGURATION==========================
    %External Functions used:
    %movestage
    %————————————————————————————————————-
    % Output Structure - handles.output.[]
    % e.g. - handles.ouput.Date
    %————————————————————————————————————-
    % .date (time stamp)
    % .image (image matrix)
```

```
% .target (Target Type): element of [2 4 6 8 10 12]
% .contrastRatio (Contrast ratio value): 0 < CR < 1
% .distance (Distance from stage to home in mm)
handles.contrast = cell(8,1);
handles.output = struct([]);
%-----------------------------------------------
% Temporary internal variables - handles.int.[]
%-----------------------------------------------
 handles.int.s1 =0; %(Distance from home of stage 1)
 handles.int.s2 =0; %(Distance from home of stage 2)
 handles.int.serialID = cportopen('com2');
 % .currentTarget %(The current target)
 handles.int.index = 1; %(Current index into the strucuture, iterates when an image is taken)
 set(handles.radiobutton1,'Value',1);
 handles.int.button = 1;

handles.int.TARGET = [0 5 10 15 20 25];
%-----------------------------------------------
%Notes
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes GatherData wait for user response (see UIRESUME)
uiwait(handles.figure1);
% — Outputs from this function are returned to the command line.
function varargout = GatherData_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
%-----------------------------------------------
%GET IMAGE CODE-------------------------------
%-----------------------------------------------
% — Executes on button press in GetImage.
function GetImage_Callback(hObject, eventdata, handles)
%Get and store the picture
tempIndex = handles.int.index;
tempImage = vcapg2;
nullImage = handles.nullImage;
SQRI= moireSqri(14,tempImage,nullImage,50,50);
handles.output(tempIndex).image = (tempImage);
handles.output(tempIndex).date = clock;
handles.output(tempIndex).currentTarget = handles.int.button;
handles.output(tempIndex).keep = 1;
```

```
handles.output(tempIndex).SQRI = SQRI;
handles.output(tempIndex).distance = get(handles.Slider,'Value');
% Calculate and store the contrast ratio
handles.contrast{handles.int.button,1} = [handles.contrast{handles.int.button,1};[get(handles.Slider,'Value'),handles.output(tempIndex).SQRI
]]
hold on
axes([handles.axes1]) % Select the proper axes
cla
%Plot the contrast ratio
set(handles.axes1,'XLim',[0 2]);
set(handles.axes1,'Units','centimeters')
set(handles.axes1,'Position',[1.295 9.994 13.246 7.244])
set(handles.axes1,'NextPlot','add')
markers = '.ox+*s';
for i = 1:8
 if ~isempty(handles.contrast{i,1})
 h = line(handles.contrast{i,1}(:,1),handles.contrast{i,1}(:,2));
 set(h,'Marker',markers(i))
 end
end
handles.int.currentLine = h;
%Write the image
axes([handles.axes2])
cla
image(tempImage)
%Update the index
handles.int.index = handles.int.index + 1;
% Update handles structure
guidata(hObject, handles);
%————————————————————————
% — Executes on button press in Undo.
function Undo_Callback(hObject, eventdata, handles)
%Needs fleshing out after doing movement code
tempIndex = handles.int.index - 1
%Set the keep value to zero
handles.output(tempIndex).keep = 0;
%Remove the previous point and replot
handles
[len,x] = size(handles.contrast{handles.int.button,1})
len = len -1;
if len==0
 handles.contrast{handles.int.button,1} =[]
 axes([handles.axes2])
 cla
end
```

```
if len~=0
 handles.contrast{handles.int.button,1} = [handles.contrast{handles.int.button,1}(1:len,:)]
end
markers = ['.ox+*s'];
axes([handles.axes1]); % Select the proper axes
cla;
for i = 1:8
 if ~isempty(handles.contrast{i,1})
 h = line(handles.contrast{i,1}(:,1),handles.contrast{i,1}(:,2));
 set(h,'Marker',markers(i));
 end
end
handles.int.index = tempIndex;
guidata(hObject, handles);
%-----------------------------------------------------------------
%MOVE 1 CODE-------------------------------------------------------
%OK
%-----------------------------------------------------------------
% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
handles.int.button = 1;
off = [handles.radiobutton2,handles.radiobutton3,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
handles.int.button = 2;
off = [handles.radiobutton1,handles.radiobutton3,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
handles.int.button = 3;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton4,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
handles.int.button = 4;
```

Section B.4 gatherData

```
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton5,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
handles.int.button = 5;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton4,handles.radiobutton6];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
% — Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
handles.int.button = 6;
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
handles.radiobutton4,handles.radiobutton5];
mutual_exclude(off)
% Update handles structure
guidata(hObject, handles);
%————————————————————————————————————
% — Executes on button press in Move1.————————————-
function Move1_Callback(hObject, eventdata, handles)
targetValue = handles.int.button
moveStageDelta(handles.int.s1,handles.int.TARGET(targetValue),1,handles)
handles.int.s1 = handles.int.TARGET(targetValue)
guidata(hObject, handles);
%————————————————————————————————————-
%MOVE 2 CODE————————————————————————————
%In Testing
%————————————————————————————————————-
function Slider_Callback(hObject, eventdata, handles)
% — Executes on button press in Move2.
function Move2_Callback(hObject, eventdata, handles)
moveStageDelta(handles.int.s2,get(handles.Slider,'Value'),2,handles)
handles.int.s2 = get(handles.Slider,'Value');
% Update handles structure
guidata(hObject, handles);
% — Executes during object creation, after setting all properties.
function Slider_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background, change
% 'usewhitebg' to 0 to use default. See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
```

```
 set(hObject,'BackgroundColor',[.9 .9 .9]);
else
 set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% — Executes on button press in Home2.
function Home2_Callback(hObject, eventdata, handles)
%Move the stage home and let the user know it is moving
set(handles.Moving2,'String','Moving.....')
stagehome(handles.int.serialID,2,-1)
set(handles.Moving2,'String',' ')
%Set the slider to the home position
set(handles.Slider,'Value',0);
%Update the tracking value
handles.int.s2 = 0;
% Update handles structure
guidata(hObject, handles);
%ADMINISTRATION CODE————————————————————
% — Executes during object creation, after setting all properties.
function SampleName_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc
 set(hObject,'BackgroundColor','white');
else
 set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function SampleName_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'String') returns contents of SampleName as text
% str2double(get(hObject,'String')) returns contents of SampleName as a double
% — Executes on button press in Finished.
function Finished_Callback(hObject, eventdata, handles)
cportclose(handles.int.serialID)
guidata(hObject, handles);
% UIWAIT makes GatherData wait for user response (see UIRESUME)
uiresume(handles.figure1);
%————————————————————————————————————-
%SUB FUNCTIONS
%————————————————————————————————————-
%————————————————————————————————————-
function moveStageDelta(presentPosition, finalPosition, button, handles)
switch button
 case 1
 %Make sure the push-button stays down
 set(handles.Move1,'Value',0);
 %Tell the user the stage is moving
```

```
set(handles.Moving1,'String','Moving.....');
%Move the stage
moveStage(handles.int.serialID,1,(finalPosition-presentPosition));
%Delete the 'Moving' text
set(handles.Moving1,'String',' ');
% Bring the button back up
set(handles.Move1,'Value',1);

case 2
%Make sure the push-button stays down
set(handles.Move2,'Value',0);
%Tell the user the stage is moving
set(handles.Moving2,'String','Moving.....');
%Move the stage
moveStage(handles.int.serialID,2,(finalPosition-presentPosition));
%Delete the 'Moving' text
set(handles.Moving2,'String',' ');
% Bring the button back up
set(handles.Move2,'Value',1);
end

%————————————————————————————————————
function mutual_exclude(off)
set(off,'Value',0);
% — Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton8 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
handles.nullImage = vcapg2;
guidata(hObject, handles);
```

5.5 createDisplay Code

```
function X = createDisplay(D,pixelOrder)
    %Make the display, should be on three seperate layers
    %gewt
    subpixel_s=[D.subpixel_s.h,D.subpixel_s.v];
    blackmatrix_s=[D.blackmatrix_s.h,D.blackmatrix_s.v];
    trans_s=[D.trans_s.h,D.trans_s.v];
    red = subpixel([1,0,0],subpixel_s,blackmatrix_s,trans_s,0);
    green = subpixel([0,1,0],subpixel_s,blackmatrix_s,trans_s,0);
    blue = subpixel([0,0,1],subpixel_s,blackmatrix_s,trans_s,0);
    if pixelOrder == 'R'
```

```
 pixel = [red green blue];
else
 pixel = [blue green red];
end
X = stamp(pixel,D.Pixels.v,D.Pixels.h);
```

5.6 subPixel Code

```
function ColourFilter = SubPixel(PixelColour,PixelSize,BlackMatrixSize,TransistorSize,show)
    %function ColourFilter = SubPixel(PixelColour,PixelSize,BlackMatrixSize,TransistorSize,show)
    %Pixel colour is vector with [R,G,B] as red green and blue components respectively
    %Pixel size is vector with [hPixelSize,vPixelSize] in 10s of micrometers
    %BlackMatrixSize is vector : [hTraceWidth,vTraceWidth] in 10s of micrometers
    %Transitor sixe is vector : [hTransistorSize,vTransistorSize] in 10s of micrometers
    %Image is plotted if show is true
    %Make Colour filter
    % close all
    ColourFilter=ones(PixelSize(2),PixelSize(1),3);
    for i=1:3
     i;
     ColourFilter(:,:,i)=ColourFilter(:,:,i).*PixelColour(i); %Make each layer of sub pixel a component of red,
green or blue
    end
    %Add horizontal matrix and vertical matrix
     %Add left ends
     ColourFilter((1:BlackMatrixSize(1)),:,:)=0;
     ColourFilter(:,(1:BlackMatrixSize(2)),:)=0;
     %Add right ends
     hEnd=PixelSize(1)-(BlackMatrixSize(2)-1);
     vEnd=PixelSize(2)-(BlackMatrixSize(1)-1);
     ColourFilter(:,(hEnd:PixelSize(1)),:)=0;
     ColourFilter((vEnd:PixelSize(2)),:,:)=0;
    %Add transistor
    ColourFilter((1:TransistorSize(1)),(1:TransistorSize(2)),:)=0;
    %Show image of pixel
    if show
     image(ColourFilter);
     axis([0,max(PixelSize),0,max(PixelSize)]);
    end
```