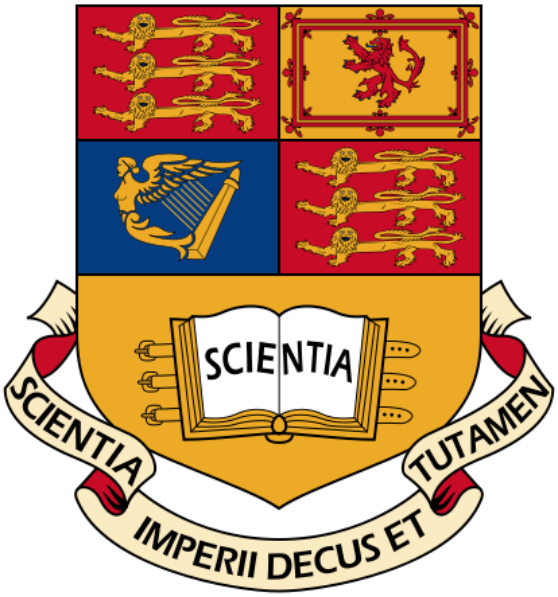
Imperial College London

Programming camera and lighting systems for controlled reflectance acquisition



Author:
Sabin Bhattarai

Supervisor:
Abhijeet Ghosh

BEng Computing

16 June 2015

Abstract

3D modelling is being used in industries like animation, gaming, filming and interior designing in order to perform dimensional and comparative analysis, it also has great significance in medical industries for interactive representations of anatomy. Real world objects behave differently in different environments due to varying colour, texture and reflectance properties, this means powerful techniques such as photometric stereo are vital tools in image processing. These tools will be a key focus in building high quality 3D reconstructions throughout this project. Although much work has been done in optimising algorithms in obtaining 3D models, rather less has been performed in creating the infrastructure required to obtain these 3D models.

Similarly various light stage systems have been built at UC Berkeley and the USC Institute for Creative Technologies over the years, enabling a variety of facial scanning, reflectance measurement and performance capture which has resulted in the creation of a few of the first photoreal digital actors. This concept of light stage will be used to create a system that allows us to gather more detailed research on image based lighting.

In this project we will look at three main stages that will create two infrastructural setup allowing to construct an application applicable for photo-metric stereo experiment. The first stage will involve the construction of an infrastructure based on light stage, for which it requires to control a camera out of the closed system. Therefore a camera will be programmed to automatically capture the data it requires. Also usage of wireless flashes acting as a light source will be examined. Secondly we will built a system that makes use of color Kinetics lights along with the use of C++ and Python programming language. From this we will build a network to control the lights via transferring of packets using various networking protocols. Controlling lights means we would have an opportunity to obtain relevant data as of photo-metric stereo technique. Lastly with all the data obtained from the previous setup and measurements, an application will be created to represent 3D model and image based lighting.

For evaluation, a setup that will precisely obtain the data sets required to create an application on images based lighting and 3D model reconstruction will be examined. Various results will be compared and analysed in accordance to various setups built. Infrastructure required to build a system will be tested and verified by the supervisor and PHD researchers. We will also demonstrate that our systems as a whole will produce expected results as compared to results obtained from higher spec systems.

Acknowledgments

I would like to thank my supervisor, Abhijeet Ghosh, for his continuous support and enthusiasm to successfully complete the project.

I would also like to thank Jeremy Riviere and Antoine Toisoul, PHD students, for providing in depth knowledge on technicalities throughout the project.

Contents

List of Figures

[1] Aydogan Akcay. Photometric Stereo yontemi.http://www.mavis.com.tr/blog/?tag=yuzey-kontrolu , May 2013. [Online; accessed 2015-01-18].

[2] Hugh Fisher. BRDF http://escience.anu.edu.au/lecture/cg/GlobalIllumination/BRDF.en.html
[Online; accessed 2015-01-20].

[3] Eric W. Weisstein. Spherical Coordinates http://en.wikipedia.org/wiki/Spherical_coordinate_system
[Online; accessed 2015-01-22].

[4] wynn2000introduction, title=An introduction to BRDF-based lighting, author=Wynn, Chris, journal=Nvidia Corporation, year=2000.

[5] Wikipedia. Specular Reflection http://en.wikipedia.org/wiki/Specular_reflection
[Online; accessed 2015-02-02].

[6] Physics class room. Diffuse Reflection http://www.physicsclassroom.com/class/refln/Lesson-1/Specular-vs-Diffuse-Reflection
[Online; accessed 2015-01-27].

[7] KMD Lighting Design LLC. Variants of Reflectors http://www.kmdlightingdesign.com/optics.html
[Online; accessed 2015-02-04].

[8] R.Woodham, Photometric method for determining surface orientation from multiple images. Optical Engineering , 19(1):139-144, 2010
[Online; accessed 2015-02-08].

[9] Kim, Hyeongwoo, Bennett Wilburn, and Moshe Ben-Ezra. "Photometric stereo for dynamic surface orientations." Computer VisionâĂŞECCV 2010. Springer Berlin Heidelberg, 2010. 59-72.

[10] USC Institute for Creative Technologies. Light Stage 1.0 http://gl.ict.usc.edu/LightStages/
[Online; accessed 2015-02-13].

[11] USC Institute for Creative Technologies. Light Stage 2.0 http://gl.ict.usc.edu/Research/LS2/
[Online; accessed 2015-02-13].

[12] USC Institute for Creative Technologies. Light Stage 3.0 http://gl.ict.usc.edu/Research/LS3/
[Online; accessed 2015-02-13].

[13] USC Institute for Creative Technologies. Light Stage 5.0 http://gl.ict.usc.edu/Research/LS5/
[Online; accessed 2015-02-13].

[14] USC Institute for Creative Technologies. Light Stage 6.0 http://ict.usc.edu/prototypes/light-stages/
[Online; accessed 2015-02-15].

[15] USC Institute for Creative Technologies. Animated Digital Human http://gl.ict.usc.edu/Research/RHL/
[Online; accessed 2015-02-19].

[16] Philips Color Kinetics http://www.colorkinetics.com/support/datasheets/iColor_MR_gen3_ProductGuide.pdf
[Online; accessed 2015-02-26].

[17] Canon DSLR SDK Europe https://www.didp.canon-europa.com/
[Online; accessed 2015-02-15].

[18] Canon Speedlite 430 EX II http://photo-tips-online.com/review/canon-speedlite-430ex-ii-flash/
[Online; accessed 2015-05-1].

[19] EOS utility Flash Settings http://thedigitalstory.com/photography/
[Online; accessed 2015-05-12].

[20] Canon camera setting and remote shooting http://www.usa.canon.com/cusa/consumer/standard_display/eos_utility?pageKeyCode=noLeftNavigation/
[Online; accessed 2015-05-21].

[21] Polarization type http://www.bigshotcamera.com/learn/lcd-display/polarization
[Online; accessed 2015-05-22].

[22] Unpolarized light to polarized light http://en.wikipedia.org/wiki/Polarization_(waves)
[Online; accessed 2015-05-26].

[23] Polarization effect http://www.physicsclassroom.com/class/light/Lesson-1/Polarization
[Online; accessed 2015-06-1].

[24] Cross polarization process http://www4.uwsp.edu/physastr/kmenning/Phys250/Lect42.html
[Online; accessed 2015-06-1].

[25] Interreflection effect http://image.slidesharecdn.com/3introlightfields-130924104409-phpapp01/95/introduction-to-light-fields-42-638.jpg?cb=1380019981
[Online; accessed 2015-04-24].

[26] PDS-70mr http://www.colorkinetics.com/ls/pds/pds70mr/
[Online; accessed 2015-05-28].

[27] PDS-70mr with 14(max)fixtures http://www.colorkinetics.com/support/datasheets/PDS-70mr_24V_Product_Guide.pdf
[Online; accessed 2015-05-17].

[28] IColor MR Gen3 http://www.colorkinetics.com/support/datasheets/iColor_MR_gen3_ProductGuide.pdf
[Online; accessed 2015-05-19].

[29] IW MR Gen3 http://www.colorkinetics.com/support/datasheets/iW_MR_gen3_ProductGuide.pdf
[Online; accessed 2015-05-19].

[30] Quickplay Pro http://www.colorkinetics.com/support/userguides/Addressing_Configuration_Guide_QPP.pdf
[Online; accessed 2015-04-8].

[31] Wireshark https://www.wireshark.org/docs/wsug_html/
[Online; accessed 2015-03-28].

[32] Buddha http://pages.cs.wisc.edu/~csverma/CS766_09/Stereo/stereo.html
[Online; accessed 2015-03-9].

[33] Santa Maria visits Light stage x http://i.ytimg.com/vi/9gTt1FohvDY/maxresdefault.jpg
[Online; accessed 2015-06-15].

[34] Realistic Graphics and Imaging http://wp.doc.ic.ac.uk/rgi/
[Online; accessed 2015-06-15].

[35] VTK http://www.vtk.org/doc/nightly/html/
[Online; accessed 2015-06-1].

[36] OpenCV http://opencv.org/
[Online; accessed 2015-06-1].

1 Introduction

1.1 Motivation

Digitally generated humans or virtual characters being able to move, speak and think has been an experimental aspect in present digital world. Realistic representation of such characters with convincingly being able to add lighting effects on the characters has become important. Identifying the underlying implementation on how such characters can be modelled and what data sets are to be captured and how to build an infrastructure to obtain those data sets has become an essential part of research in recent years. Thus the project will go onto describing the techniques used for geometry orientation capture, realistic rendering, imaged based lighting and various setups required to obtain realistic computer graphics application.

As discussed, computer based image understanding requires a larger set of data from an image. When an object is under consideration for 3D reconstruction, knowing the orientation of the object at each point becomes important. Photometric stereo technique thus becomes an important aspect in understanding the orientation of object which requires rapid reflectance measurements. Similarly by using photometric stereo technique, one can experiment with reflectance acquisition examples such as measuring diffuse and specular albedo (total reflectivity) as well as surface normal maps (shape information).

The key idea is therefore to get familiarised with the concept and techniques involved in image processing and most importantly build a programmable reflectance acquisition setups to obtain as much relevant data required in computer based renderings. Also with many existing setups such as Light stages used for image processing, our goal becomes to make as much research to obtain higher understanding on the underlying implementation and scale it up in near future.

1.2 Objectives

The overall aim of this project is to build a programmable reflectance acquisition setups using DSLR and/or machine vision cameras and controllable light sources for realistic computer graphics applications. The camera light source systems will include DSLR cameras and flashes from Canon. For the controllable LED light source, LED lights from Philips Colour Kinetics will be used.

As a first step, the aim is to employ the Canon SDK to control DSLR cameras and flashes (via wireless protocol) for rapid reflectance measurements. Similarly programmable network of LED lights from Philips Colour Kinetics for reflectance measurement is to be built using Python/C++ based API to control the LED lights. It also involves extending the network to control RGB and white LED lights over an Ethernet network.

With the setup up and running, an important experiment to examine would be understand the differences between reflectance acquisition with broad spectrum white illumination and narrow band RGB illumination and/or combine both sources of illumination for multispectral imaging of material reflectance including faces.

Hence to achieve the above mentioned objectives it becomes important to do background research on photometric stereo to understand an objects orientation, photometric stereo with narrow band RGB illumination, light stages and most importantly the setups required to obtain relevant results.

2 | Background

2.1 Photometric Stereo

Photometric stereo[13] is a technique that obtains high quality image based 3D reconstructions. The technique involves taking a sequence of images usually three or more by varying the direction of incident illumination, while keeping the same viewpoint as shown in Fig 2.1. With the intensity variation observed in each pixel one can make an estimation of the local orientation of the surface onto that pixel as the imaging geometry remains constant[5]. Once the surface orientations are observed, it becomes easier to integrate these surface orientations to get the surface geometry of an object. The idea therefore is to get the orientations of vectors that will perpendicularly fall to the surface (object of interest) at multiple points. Thus by taking the increasing number of points in the surface and normal vectors, the quality of the map can be improved.

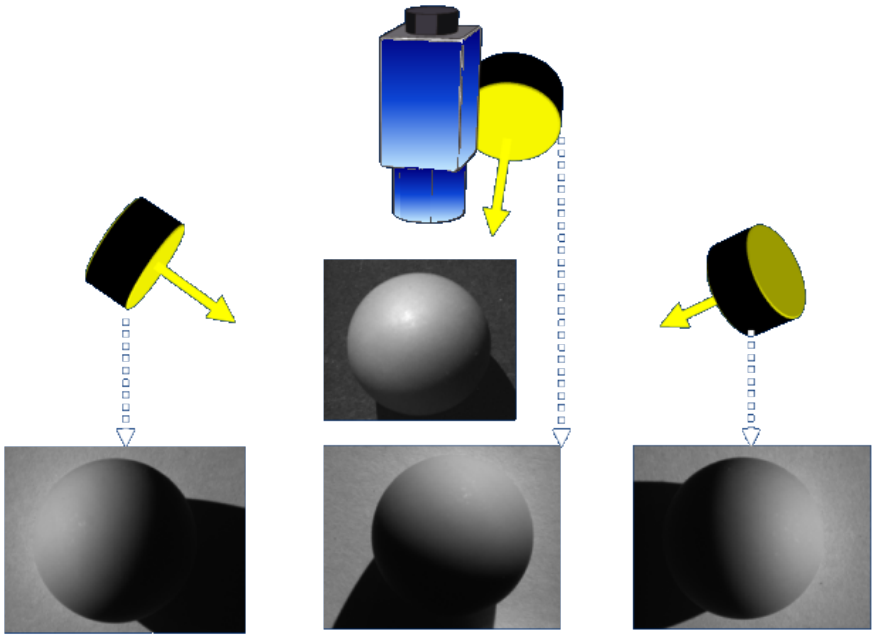


Figure 2.1: Photometric stereo setup under one viewpoint and varying light source [1]

As shown in figure 2.1 we have three images obtained using one camera but different lighting conditions. While clearly the same underlying surface is being portrayed, the

detailed patterns of brightness are different i.e. some areas on one picture has higher contrast while same area on another picture is not visible. This shows the image of a three dimensional object depends on its reflectance properties, its shape and the distribution of light sources.

Let us take an example on the kind of output we expect when performing the photometric stereo experiment. The result expected is a normal map, albedo, height map and a 3D mesh. The following shows the input (including light sources) and output images for a rock whose surface orientation has been deduced from the photometric experiment as shown.

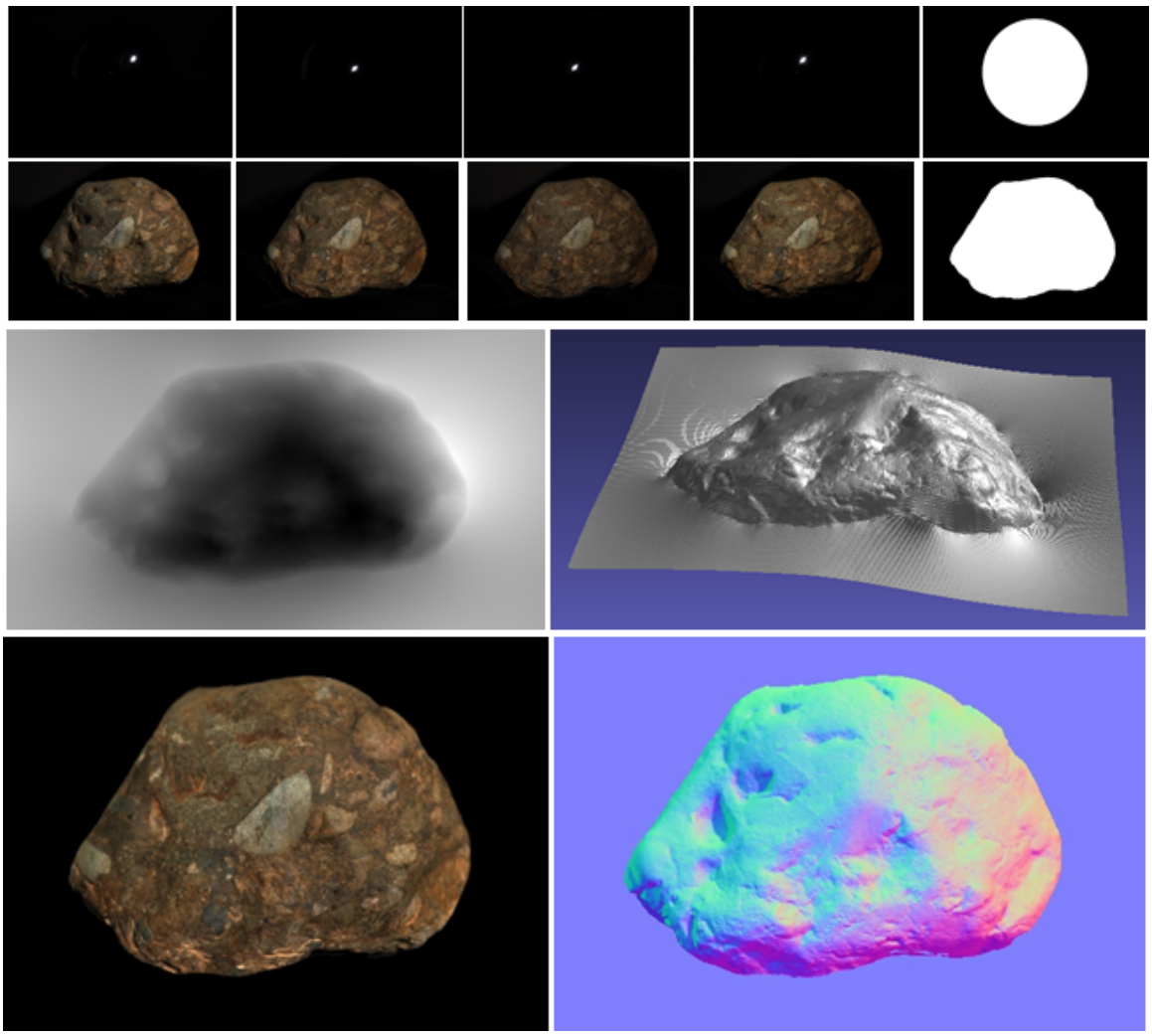


Figure 2.2: Photometric stereo experiment performed for Rock object

Similarly, photometric stereo provides the surface reflectance properties as part of the same experiment. The method makes use of the so called **reflectance maps** in the form of look-up tables. One of the ways in obtaining the tables is to calibrate sphere of the same material as that of the imaged surface, allowing obtained intensities

to be directly mapped to surface normal. The term photometric relates to the use of radiance values at single image location. **Radiance** and **Irradiance** being the measure for lighting intensities, allows to indicate how much of the power a reflecting surface will be received by an optical device observing at the surface from an angle of view. This concept therefore becomes important for photometric stereo technique in obtaining light intensities equations for further calculations. The mathematics for photometric stereo involves complex vector analysis, advanced calculus along with equating differential equations which will be discussed briefly. We therefore make few assumptions to simplify calculations i.e. the surface reflectance model is assumed to be **Lambertian**, surface albedo is already known for each point on surface and the light sources are point light sources at infinity.

Despite its popularity in applications like facial recognition and industrial product quality assurance/ quality control, photometric stereo has two main limitations

- One needs to keep the 3D scene as static as possible during the illumination changes. This limitation therefore does not allow reconstruction of deformed 3D objects.
- Second limitation involves images to be taken from single viewpoint. This thus avoids full 3D surface reconstruction

Thus a setup will be discussed which tries and overcome these limitations. Photometric stereo with coloured light variant will be introduced. It will be used to obtain separate reconstruction of object for each gathered photograph. The required setup will consist of video camera with three coloured light sources (red, green and blue). Observation will be done in an environment where red, green and blue light emitting from different direction towards the object with Lambertian surfaces.

The general idea on this part would be to use the photometric setup to take three different images, calculate the reflectance map using the BRDF function keeping in mind the assumption of Lambertian surfaces. The calculated reflectance map will then be used in the formula discussed below to get the surface normal for each point in the Lambertian surface. Once the normal is deduced we will use the integration rule to obtain the 3D orientation of the object[10]. This concept will be briefly explained in this order.

2.1.1 BRDF

A mechanism that plays an important role in recognising interactive photorealism is the idea of Bi-directional Reflectance Distribution Function (BRDF)[9]. BRDF depends on the light under consideration, the positional variance and importantly to the incoming and outgoing direction. BRDF in functional notation can therefore be represented as

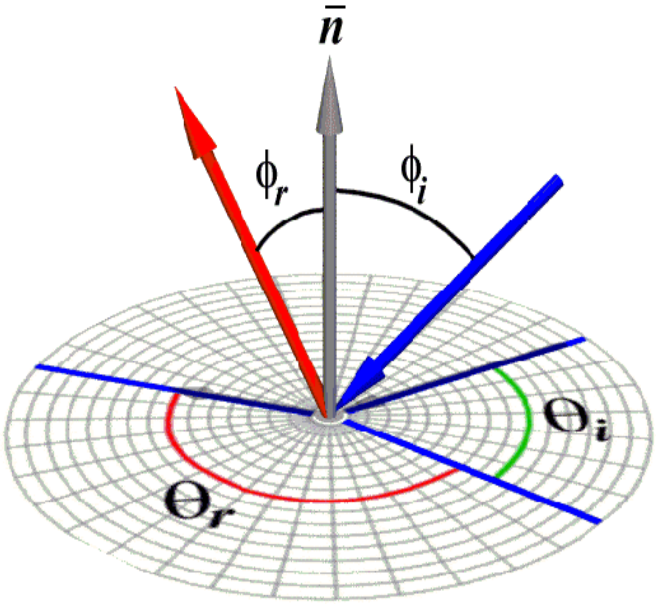


Figure 2.3: BRDF relating light incident and light reflected [2]

$$\text{BRDF } \lambda(\theta_i, \varphi_i, \theta_o, \varphi_o, u, v)$$

where θ_i, φ_i represents the incoming light direction in spherical coordinates, while θ_o, φ_o denotes the outward reflected direction in **spherical coordinates**. Similarly λ defines the wavelength under consideration and u, v relates to the surface position in texture space. The BRDF has units sr^{-1}, with steradians (sr) being a unit of solid angle.

Excluding positional variance becomes possible when only homogenous materials are taken into account and an assumption is made that the reflectance properties of a material do not vary with the spatial position. This allows BRDF functions to not include the positional variance defining itself as function of incoming and outgoing directions along with the wavelength i.e.

$$\text{BRDF } \lambda(\theta_i, \varphi_i, \theta_o, \varphi_o)$$

Similarly λ being the wavelength and the fact that BRDF depends on the wavelength or colour channel, it becomes convenient to obtain the BRDF value separately for each colour channel (i.e. Red, Green and Blue) omitting the use of λ in the function[14].

The BRDF function thus allows to generate the reflectance property by integrating over the specified **solid angles**, for a defined incident and reflected ray geometry. It is therefore necessary to understand the underlying definition on spherical coordinates and solid angles which is defined briefly below.

2.1.1.1 Spherical Coordinates

As opposed to Cartesian coordinates spherical coordinates is specified for three dimensional space with position defined by radial distance($\mathbf{r}$), polar angle θ and azimuthal angle φ as shown in the figure.

Often we relate vectors to Cartesian-space vectors as $v = (v_x, v_y, v_z)$ which brings some complications while dealing with parameterized BRDFs. It therefore becomes useful to take spherical coordinate system for representing vectors when BRDF function is taken into consideration. In spherical coordinates, a vector will therefore be defined as magnitude,(ρ), and a pair of angles relating to the difference in two reference basis vectors.

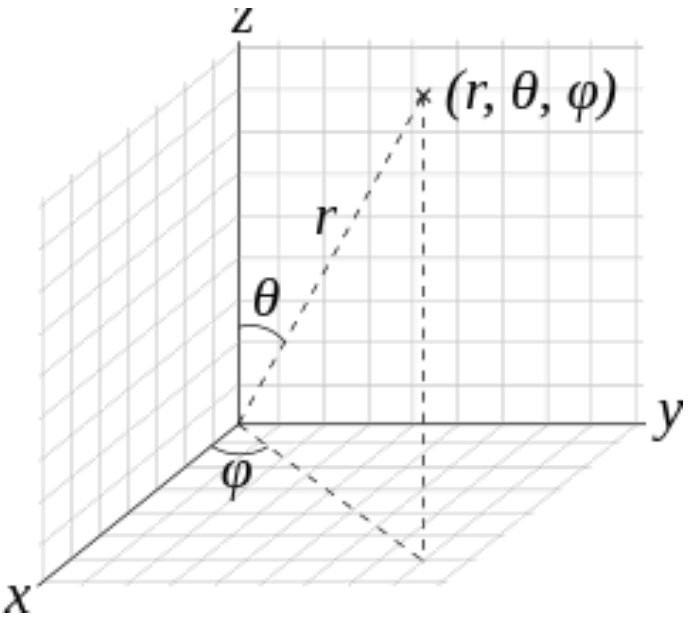


Figure 2.4: Spherical coordinates (r, θ, φ): radial distance r, azimuthal angle θ, and polar angle φ. [3]

It is possible to form a relationship between Cartesian coordinates (v_x, v_y, v_z) and spherical coordinates (θ, φ) when normalized direction vector $(1 = \sqrt{(v_x^2 + v_y^2 + v_z^2)})$ is considered.

$$v_x = \cos\varphi \sin\theta, \tag{2.1}$$

$$v_y = \sin\varphi \sin\theta, \tag{2.2}$$

$$v_z = \cos\theta \tag{2.3}$$

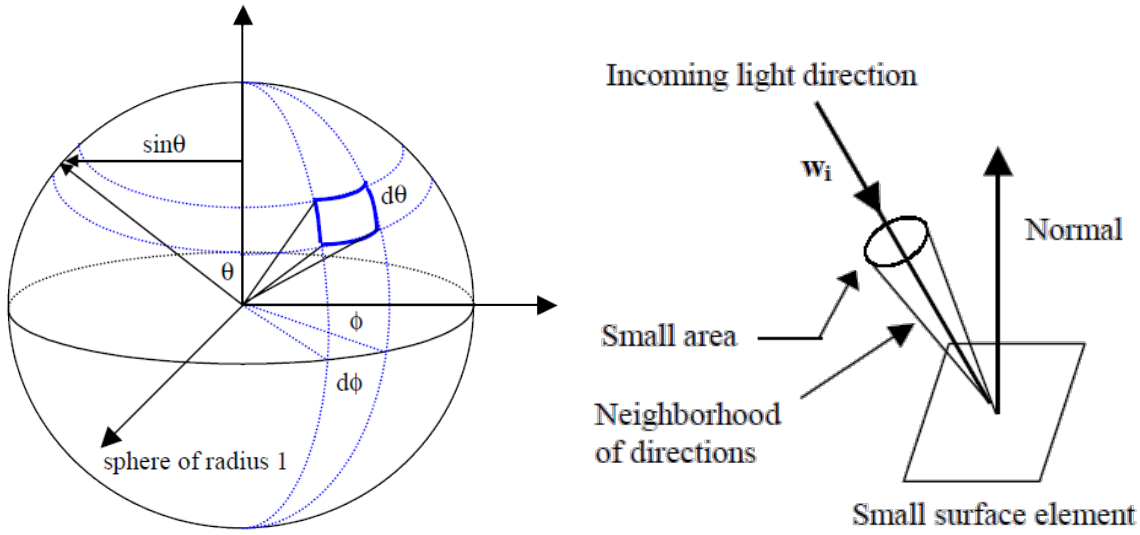
Hence it becomes easier to represent a direction with only two parameters in spherical coordinates. Since direction can be represented in the form of spherical coordinates, the BRDF function can be treated as wavelength dependent 4 dimensional function as discussed before.

2.1.1.2 Solid Angles

An object that covers certain fraction of the sphere when seen by an observer at the sphere's centre is called solid angle [14]. When dealing with BRDFs, it becomes necessary to understand how much of light arrives or leaves the surface element from

a particular direction with a view from various viewing positions. This brings up a notion of differential solid angle. It becomes important to consider how much of light is flowing through an area of space rather than considering the amount of light from a single incoming direction.

Figure 2.5 (a) below emphasizes on considering the flow through a neighbourhood of directions when determining how much light arrives or leaves the surface. It puts its basis on the fact that light is always measured in terms of energy per unit surface area. For complete understanding, it can be thought as a small rectangular region on a unit sphere as shown in figure 2.5 (b). The figure also shows a pyramid formed on the inside of the sphere which represents a volume of direction. The differential solid angle is defined as the small path that can be viewed in the sphere's surface formed by an intersection of the pyramid and the unit sphere portion.



(a) Solid angle is the area of the small patch on surface of sphere

(b) Energy per unit area, from neighborhood of directions

Figure 2.5: Solid angle representation[4]

As we discussed the notion of spherical coordinates earlier, we can now define the direction in terms of spherical coordinates as (θ, φ). Similarly we define the small differential angular changes as $d\theta, d\varphi$. With these data in hand we can obtain the differential solid angle as

Differential solid angle = height * width
$= (d\theta) * (\sin\theta * d\varphi)$
$= \sin\theta * d\varphi * d\theta$

The obtained differential solid angle will thus have a unit of $radians^2$ also known as steradians (sr).

2.1.2 Reflectance Map

Reflectance map [4] gives the association between the gradient space parameters and brightness. For many surfaces, the surface orientation solely depends on the section of the incident illumination reflected in particular direction. Reflectance properties of these surfaces can be characterized as a function of incident, emergent and phase angles. The incident and emergent angles are relative to the surface normal. The function mentioned above is related to the bidirectional reflectance distribution function (BRDF) which is the ratio of surface radiance to irradiance measure per unit surface area, per unit solid angle in the same direction of the viewer. As described above, reflectance map gives scene radiance as a function of the gradient which can be calculated from the bidirectional reflectance-distribution function (BRDF) and the distribution of source radiance.

Following describes few of the concepts that are necessary in understanding the properties and calculation of reflectance map.

2.1.2.1 Radiometry

Light can be described as photons moving at the speed of $3 * 10^8 km/s$ in the air. Each photon has an energy that relates to its wavelength and colour. Radiometry[6] is thus the science of measuring photons or electromagnetic radiation in any area of the electromagnetic spectrum. The power of electromagnetic radiation is termed as radiant flux. Radiant flux that is reflected, emitted, received or transmitted in a direction per unit solid angle by a surface is defined as radiance divide irradiance.

Radiance can be used to characterize the reflection or diffusion of incident light from any surface. The SI unit of radiance is watt per steradian per square metre (Wsr^{-1}/m^2) and is key in determining how much of the power will be received by a system observing at the surface from some angle of view. In other words, radiance is the amount of light radiated from a surface. Its complex concept derives from the fact that surface are liable of radiating into whole possible directions and at the same time can radiate different amounts of energy in different directions.

Similarly intensity or brightness of a surface is determined by how much of energy is being received by the imaging system per unit apparent area i.e. the surface area multiplied by the cosine of the angle between a perpendicular to the surface and the direction specified. It will soon be apparent the importance of intensity measurements when we evaluate reflectance map for a surface to determine its orientation.

2.1.2.2 Reflectors

Surfaces reflect light in accordance to the law of reflection, making it possible to determine angle of incidence when a normal to the surface is known. Reflectivity of surface thus is the ratio of reflected power to incident power and relates to refractive index of a material alongside the electronic absorption spectrum[6]. Reflectors can be categorised as

1. **Specular Reflection** A mirror-like reflection of light from a smooth surface, where incoming light from a direction is reflected to a distinct outer direction. Specular reflection does not necessarily reflect all of the incident light as some materials can absorb and transmit lights through the surface. Polished metallic objects can be a perfect example of specular reflection.

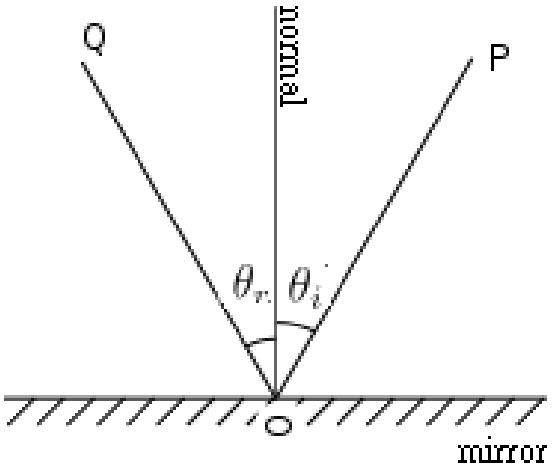


Figure 2.6: Diagram on Specular Reflection[5]

The reflected rays becomes parallel to the surface if the incident rays are placed parallel to the surface allowing us to understand the direction of reflection. Direction is determined by vector of incidence and surface normal vector i.e. assume an incident direction (d_i) hits the surface with surface normal direction (d_n) and we require to deduce the specular reflected direction (d_s). So formulation to determine the d_s would be as

$$\hat{d}_s = 2(\hat{d}_n \dot{} \hat{d}_i) * \hat{d}_n - \hat{d}_i$$

where $\hat{d}_n \dot{} \hat{d}_i$ refers to the dot product between normal and the incident direction vector. Also note that unit vectors are taken for each vector which can be calculated as

Unit vector of $d_n = d_n / \|d_n\|$ i.e. if $d_n = (x, y, z)$ we would get $\|d_n\|$ by $\sqrt{x^2 + y^2 + z^2}$

2. **Diffuse(Lambertian)** In diffuse reflection, we have surfaces that reflects the incident ray in many angles. This is because of the molecules that make up the surface have different reflecting property. Although each ray falling on the surface follows law of reflection, we might think how a rough surface diffuses the beam of light. This is because each ray falling on the rough surface do follow the law of reflection but each ray meets the surface at different orientation.

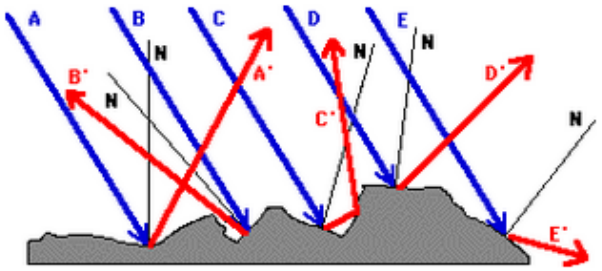


Figure 2.7: Law of Reflection with incident rays (labeled A, B, C, D, and E) [6]

As in the figure, we can observer A,B,C,D,E fall onto the surface with different normal and its reflecting behaviour. Hence different normal line causes the ray to reflect differently.

However, we are interested in Lambertian reflectance which is the property that defines the diffusely reflecting surface. In Lambertian surfaces the brightness observed from any angle of view will be constant i.e. the surface will have same radiance at all view angles. This is because the Lambertian surfaces has luminous intensity following lambert's cosine law i.e. the reflected energy from a surface area in a specific direction becomes proportional to the cosine of the angle between the direction and the surface normal.

Throughout the photometric stereo setup we will be using Lambertian surfaces. However, there are few other types of reflectors that has some similarities with either specular or diffuse reflections as presented in the figure below.

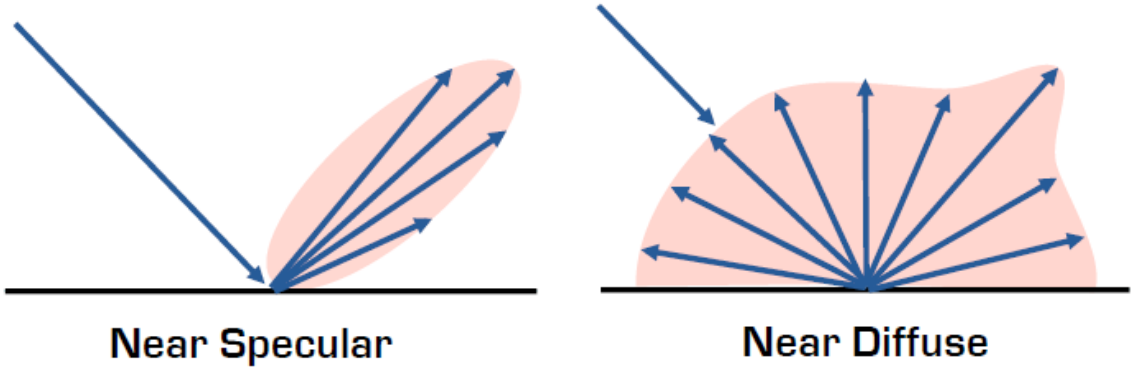


Figure 2.8: Variants of Reflectors[7]

Near specular can be defined as reflection of light from a surface to a distinct area but not in multiple directions. On the other hand near diffuse is where surfaces reflects the incident ray in many angles equally but the intensity in some areas is higher compared to other reflected rays.

2.1.2.3 Formulation

Now that we have general idea on what reflectance map is and essential terminologies necessary to describe the Reflectance, we can now derive a formula in above described terms.

Thus reflectance map can be defined as

$$R(p,q) = \rho \frac{n.s}{|n||s|}$$

Where ρ is the albedo with s defined as $[S_x, S_y, S_z]^T$ which gives a relation between light direction and viewing direction. Furthermore n is the surface normal.

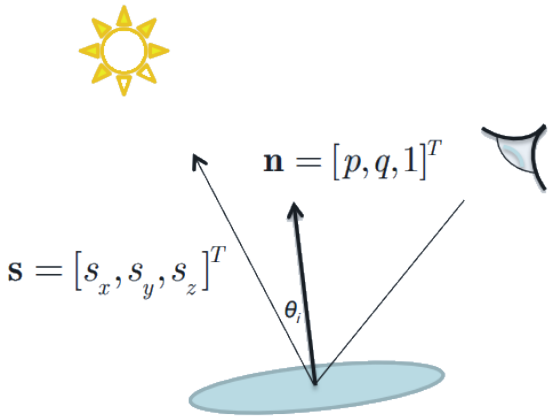


Figure 2.9: Reflectance properties with normal and incident light[8]

As from figure 2.9 we can rewrite the equation above considering s as a unit vector

$$R(p,q) = \rho \frac{s_x p + s_y q + s_z}{\sqrt{p^2 + q^2 + 1}}$$

This is called the reflectance equation, and is a second order equation of p and q. In order to visualise the function R it can be drawn as a series of contours of $R(p,q) = constant$. The questions here arises on how do you determine p and q. Although a unique solution could be determined for surface orientation at each point, given two images with different lighting conditions, an assumption has to be made that these equation were linear and independent with known albedo (ρ). However, while resolving the equations of this manner we encounter non linear equations having either no solutions or several solutions.

Hence, we will use the photometric stereo technique to deduce a simple equation that allows calculation of p, q. The next section on surface normals will show detailed process on calculating p, q. There we will also make extensive use of reflectance map equation $R(p,q)$ to deduce orientation of an object.

2.1.3 Surface Normal

A directional vector that is orthogonal to the tangent plane to a surface at a point is defined as surface normal. In order to recover the 3D structure of an object through photometric means, we require to perform normal map integration. Hence obtaining the surface normal vector at each point becomes a necessary calculation to make. In order to obtain the unit surface normal vector (n) we deduce it in terms of p and q discussed in formulation part in reflectance map above.

We know, for a given surface patch, the slopes along the x and y axis are

$[1, 0, \frac{\delta Z}{\delta X}]^T$ and $[0, 1, \frac{\delta Z}{\delta Y}]^T$ respectively.

Let us thus define p = $-(\frac{\delta Z}{\delta X})$ and q = $-(\frac{\delta Z}{\delta Y})$

This allows us to obtain the above slope vectors as $[1, 0, -p]^T$ and $[0, 1, -q]^T$. Hence the unit surface normal vector (n) can be obtained by doing the cross product between the two vectors producing

$n = [1, 0, -p]^T \times [0, 1, -q]T$

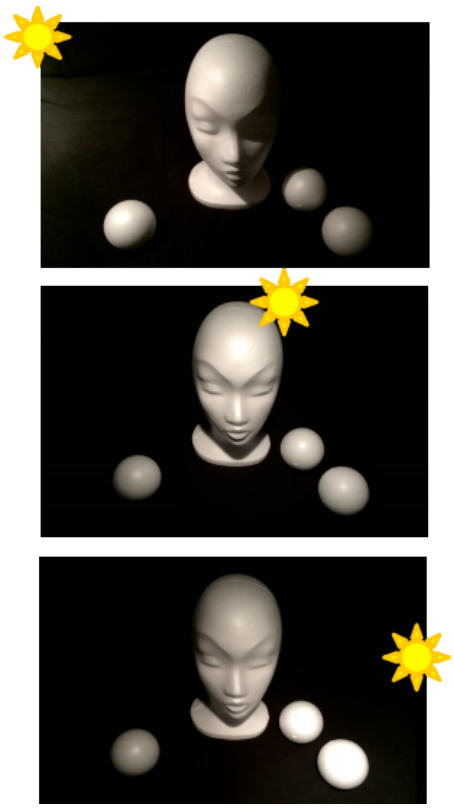
$$n = \frac{1}{\sqrt{1 + p^2 + q^2}}[p, q, 1]^T$$

Now that we have surface normal (n) in terms of p and q. We try and evaluate p and q from the photometric equation technique (assuming object as Lambertian) that involves

- Three light sources with fixed camera
- Three different images under consideration
- Each lighting condition gives $R(p,q)$ i.e. the reflectance map for each pixel.

- Depth can now be derived by integrating p and q along the surface

A simple implementation can be seen in the figure below with a setup capable of deducing three different equations involving p and q.



$$R_1(p,q) = \rho \frac{s_{x1}p + s_{y1}q + s_{z1}}{\sqrt{1 + p^2 + q^2}}$$

$$R_2(p,q) = \rho \frac{s_{x2}p + s_{y2}q + s_{z2}}{\sqrt{1 + p^2 + q^2}}$$

$$R_3(p,q) = \rho \frac{s_{x3}p + s_{y3}q + s_{z3}}{\sqrt{1 + p^2 + q^2}}$$

Figure 2.10: Reflectance equation for three differently illuminated images[8]

From the three equations above we can take different ratios e.g. $\frac{R_1}{R_2}, \frac{R_3}{R_2}$ to cancel out the unknown albedo (ρ) and form a set of linear equations for each pixel. Hence for each pixel we can now substitute to the equation below to obtain surface normal (n). i.e.

$$n = \frac{1}{\sqrt{1 + p^2 + q^2}}[p, q, 1]^T$$

2.1.4 Normal map integration

Finally with surface normal map deduced, it becomes easier to integrate the normal map. There are very many ways proposed onto performing the integration as presented by Petrovic [11]. However a simple solution becomes integrating the normal map with an iterative method. This is believed to conserve the integrability constraint, as proposed by Basri and Jacobs[1]. For this we work out the depth value of an object at the pixel position (p,q) using an Gauss-Seidel scheme of iteration until a range value K. i.e. the range is within 1 to K.

The following equation is used to obtain depth z for a given normal map (n)

$$z_{p,q}^{k+1} = \frac{1}{4}[z_{p+1,q}^{k} + z_{p-1,q}^{k} + z_{p,q+1}^{k} + z_{p,q-1}^{k} + n_{p-1,q}^{x} - n_{p,q}^{x} + n_{p,q-1}^{y} - n_{p,q}^{y}]$$

2.2 Photometric Stereo with narrow band RGB illumination

So far we have considered photometric stereo as one of the technique that reconstructs surfaces from images of an object. We figured, how by observing the altered intensity throughout the object surface under changing illumination, photometric stereo deduces the local surface orientation. Similarly we realised a way to integrate the field of local surface orientation to construct 3D shape. However the difficulty arises when we try to apply the technique to deforming objects. This is because the technique requires changing of light source direction for each captured image while keeping the object motionless. This becomes impractical when dealing with moving objects. Thus we will introduce multispectral lighting technique which captures three images in a single snapshot where each image corresponds to different lighting direction.

2.2.1 Structured and Coloured Lights

For photometric stereo with narrow band RGB illumination, we need to recognise the camera and light spectral characteristics [11]. As we are to setup a system with LED's positioned and filtered in a way that camera receives distinct Red, Green, Blue lights and not an overlapping LED spectra, we require the camera relative spectral response as shown in the figure.

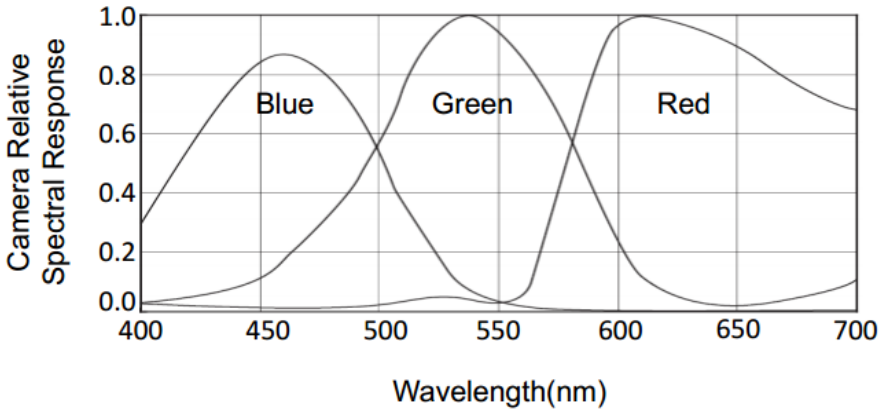


Figure 2.11: Relative spectral power distributions for different colour LED's[9]

2.2.2 Possible setup and measurments

The setup therefore involves a colour video camera with three filtered light sources i.e. with red, green and blue filters. Camera and the lights are to be placed at around 2m away from the object of interest while placing at an angle of about 30 degrees but not collinear.

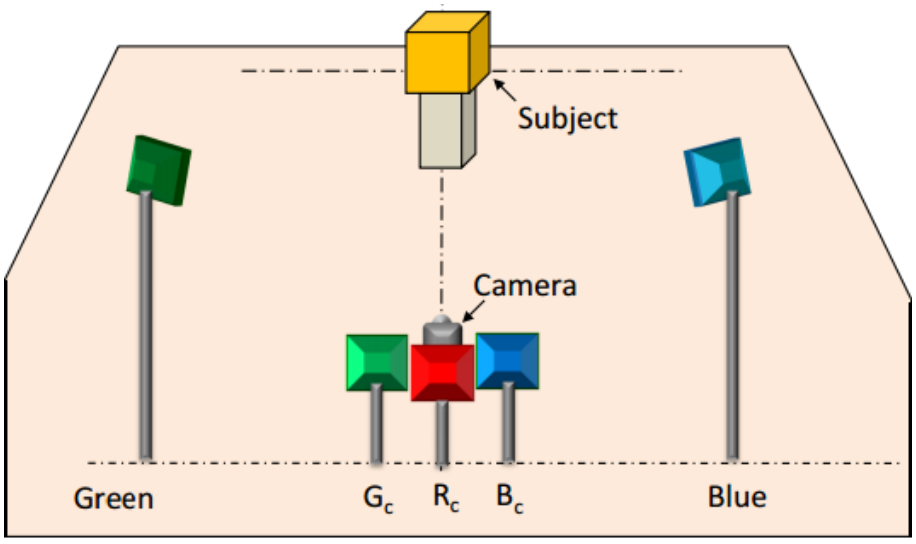


Figure 2.12: Schematic diagram representing photometric Stereo for Dynamic surface orientation [9]

As discussed in [3] an estimation can be made and by inverting the linear mapping, a link from RGB values to surface normals is obtained. Each normal map can be integrated using the Successive Overrelaxation solver(SOR) [7] to get the depth map.

2.3 Light Stages

Light stages [2] is a cage like setup with use of instruments consisting of structured light and multiple camera to capture texture, reflectance and motion of objects, mostly human faces. It not only provides an ability to simulate intensities, mixture of colours and direction of illumination over wide range of directions, but also have a degree of control over the illumination. The fact that light reflecting off the air-to-oil holds its polarization while the light through the skin loses its polarization led to the initial approach in capturing the reflectance field for human face. The very concept was further used by Paul Debevec to build several variant of light stages.

2.3.1 Light Stage 1: Acquiring Reflectance Field of Human Face

Image based lighting (IBL) allows us to simulate synthetic objects with real light. The idea behind the Light stage 1 was to apply IBL to real world objects unlike computer generated models. For this we capture the light by taking the series of images of the mirrored ball which records the colour and intensity of the light coming from every direction. We then use the global illumination algorithm to simulate the captured illumination falling on synthetic objects. When the light from imaged based lighting environment is simulated with global illumination, we see the rendered object as if the object was illuminated by real world lighting. This technique therefore becomes useful when creating the animation rendering with natural light.

Light stage 1 simulated the person's appearance in any form of illumination by forming the special lighting stage to illuminate the person from every possible direction. As shown in Figure 2.13, Light stage spiralled a single incandescent spotlight around a person's face while the digital camera recorded the face 4D reflectance field. In the course of minute the spiralling light source illuminated the face from 2000 directions. The set of images obtained could then be scaled according to the intensity of the corresponding direction of the light and its colour in the scene. For example, to generate

Figure 2.13: Light Stage 1.0[10]

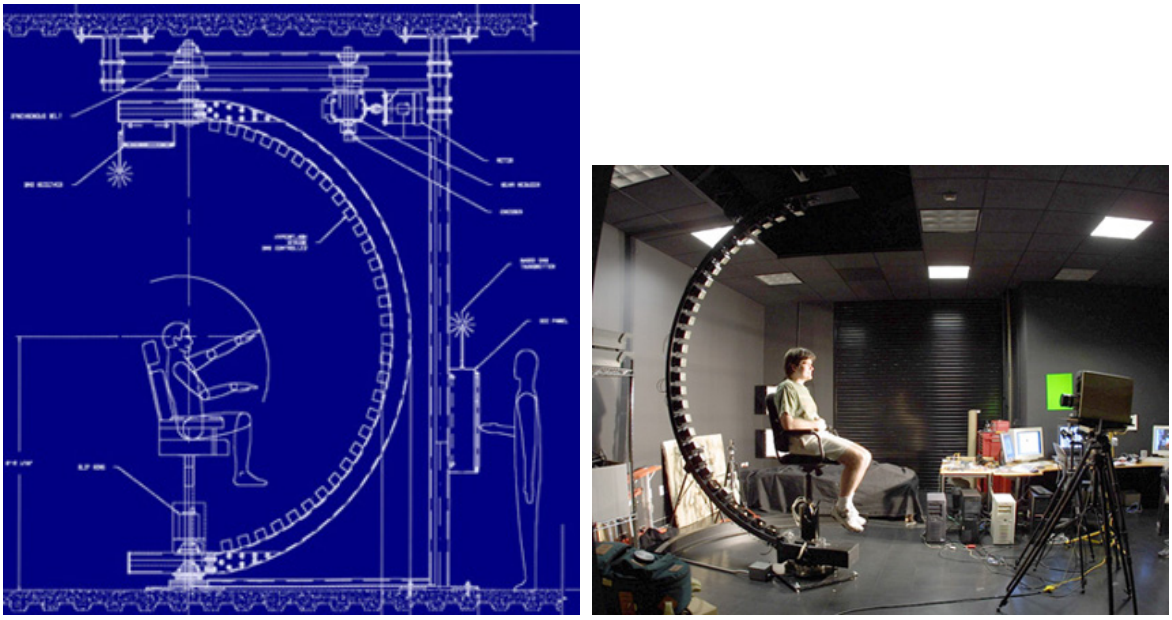
an image of the face illuminated by the lighting environment, we resample the lighting into the same set of direction and scale each image of the face by the colour and intensity of the environment in the corresponding direction. Adding this scaled images together then generates the image of the face under the sampled illumination.

In simple words, suppose we photograph a person's face under two forms of illumination, one from left and the other from right. Considering the additive property of light, we can reproduce images under both light sources by adding both images together. Furthermore we can synthetically adjust the colour and intensity of each light source by scaling the colour channels of the two images. This concept can therefore be used in larger scale to model the lighting effect on objects.

Similarly, Light stage 1 had the capability to create relightable 3D models of people's faces, while also allowing it be rendered from different viewpoints. The geometry was obtained by placing the light video projector to the Light Stage system and the reflectance estimation algorithms were used. With the reflectance function applied on each pixel in the image and making use of video projection and the reflectance estimate algorithm, it was possible to obtain reflectance maps of the face's specular intensity, diffuse colour and eventually surface normal. Furthermore the obtained maps were used in traditional computer-graphics rendering process.

2.3.2 Light Stage 2: Faster Capture and Facial Animation

Light stage 2.0 is an evolved version of the first light stage where realistic computer models of human faces are captured. It consists of arrays of lights and high-speed cameras which makes recording the reflectance of face faster by reducing the acquisition time to few seconds. It includes set of thirty-two sequential strobe-lights on a semi-circular arm which rotates around the actor in 8 seconds while the person's appearance is recorded by synchronized high-speed video cameras. It records the transformation of incident light into radiant light by a person's face while also capturing the variety of expressions that a person makes allowing to infer the appearance of a fully animated characters. The setup can be seen in the figure below.



(a) Design (b) stage implementation

Figure 2.14: Light Stage 2.0[11]

Multi-view relightable images thus obtained from many different expressions allowed to perform blends between the expressions resulting in realistic character with facial wrinkling and also dynamic reflectance. This characteristics became useful in various applications and was used by Sony Pictures Image-works to record actors Alfred Molina and Tobey Maguire for their digital stunt doubles in the movie Spider-Man 2 (2004) , Brandon Routh for the film Superman Returns (2006) , and by Weta Digital to record actress Naomi Watts for the movie King Kong (2005).

Light Stage 2.0 therefore gave an opportunity to perform basic research in human facial reflectance. It also allowed live action composition of actors into computer-generated environments along with enhancing the technology involving virtual actors. Although the full measurement of field of reflectance was not achieved at early Light stages, future versions tend to observe reflectance properties in greater details when experiments were performed involving directional polarized light.

2.3.3 Light Stage 3: Lighting Reproduction

It soon became practical to surround an actor with colour-controllable LED light sources in order to replicate the real-world or virtual lighting environment. It allowed the colours and intensities of the illumination to be calculated in an imaged based lighting environment.

The light stage 3 is a two meter sphere of inward-pointing RGB light emitting diodes i.e. 156 iColor MR red-green-blue LED light sources focused on the actor. This produces the HDRI lighting conditions on actors giving a flexibility to composite into real scenes with matching illumination. The process used in obtaining the accurate maps in this stage setup was infrared matting unlike green screens which had a disadvantage of spilling green light onto the actors. Hence, the visible-spectrum illumination on the actor was not to be affected during the lighting reproduction. Therefore an implementation involving a digital two-camera infrared matting system that composited the actor into the background environment was constructed. The system was calibrated well to match the colour illuminated by the environment and the systems colour response. Thus a moving camera composites of actors into real-world with correct illumination could be built with light stage 3.

Figure 2.15: Light Stage 3[12]

2.3.4 Light Stage 4: Concept

Light stage 4 was a conceptual idea which was able to control the environmental illumination on actors in small sets. This however was never built.

2.3.5 Light Stage 5: Performance Relighting

The main idea behind light stage 5 was providing flexibility to design and modify lighting and reflectance of an actor in postproduction. Light stage 5 consisted of 156 white LED lights which had a similar structure that of Light stage 3. With the availability of bright white LED's it was possible to record reflectance properties of actors dynamically. A sequence of time-multiplexed based lights were illuminated on the subject and high speed video camera were used to record the desired images on all conditions at a chosen frame interval.

Thus with the acquired data from the setup it was possible to find an estimation of time varying surface normal, albedo and ambient occlusion. These estimation values could then be used in transforming the actor's reflectance for stylistic effects.

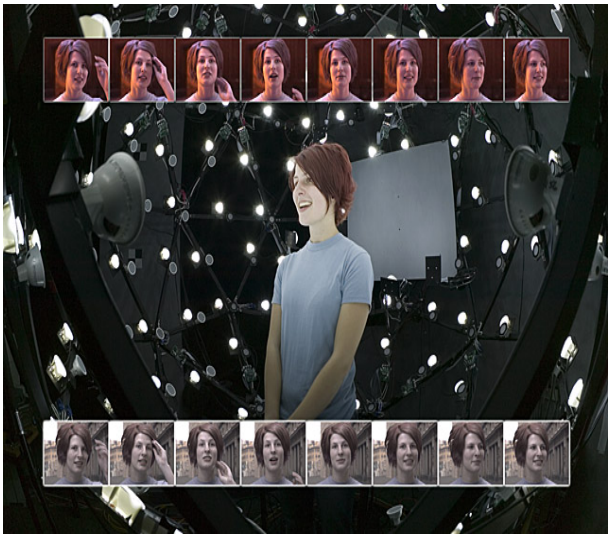


Figure 2.16: Light Stage 5[13]

Light Stage 5 on the other hand could be used for recording reflectance field similar to the traditional (one light at a time) approach. This flexibility allowed production of various applications i.e. Sony pictures Imageworks used it to digitize the reflectance of actors Will Smith and CharlizeTheron for the movie *Hancock (2008)* and a silicone maquette portraying Brad Pitt as an old man for *The Curious Case of Benjamin Button(2008)*[2].

2.3.6 Light Stage 6: Re-Lighting the While Body

With enough space available to built a larger stage, Light stage 6 was introduced for being able to illuminate the whole human body. It was just a larger version of light stage 4 mentioned earlier. It consisted of 6,666 LumiLed's Luxeon V LEDs controlled in

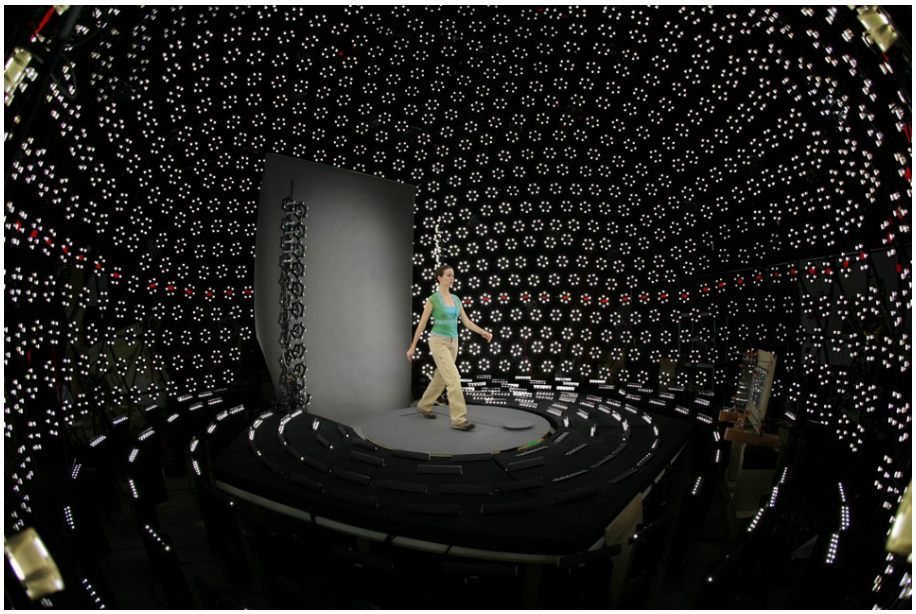


Figure 2.17: Light Stage 6[14]

groups of 6[2]. It was 8m in diameter with the floor populated with its own set of Lambertian-distribution light sources.

Light stage 6 recorded actors on a rotating treadmill at 990 frames per second. With the data received from a surrounding set of video cameras and photometric stereo, actors geometry was derived. Also Light stage 6 has recently been augmented with a Laser scanning system from ICON imaging studio to obtain the geometry and reflectance of actors such as Michael Caine and Dwagne Johnson for *Journey 2 : The Mysterious Island (2012).*

Figure 2.18: Animated digital humans rendered from arbitrary view-points and illumination from light Stage 6 data.[15]

2.3.7 Light Stage X

Light stage X [12] is a facial scanning system using programmable light sources and interchangeable filters which is able to create intensity, direction, colour and polarization of light. The following shows an image when HuffPost Science Correspondent Cara Santa Maria visits the Light Stage X.

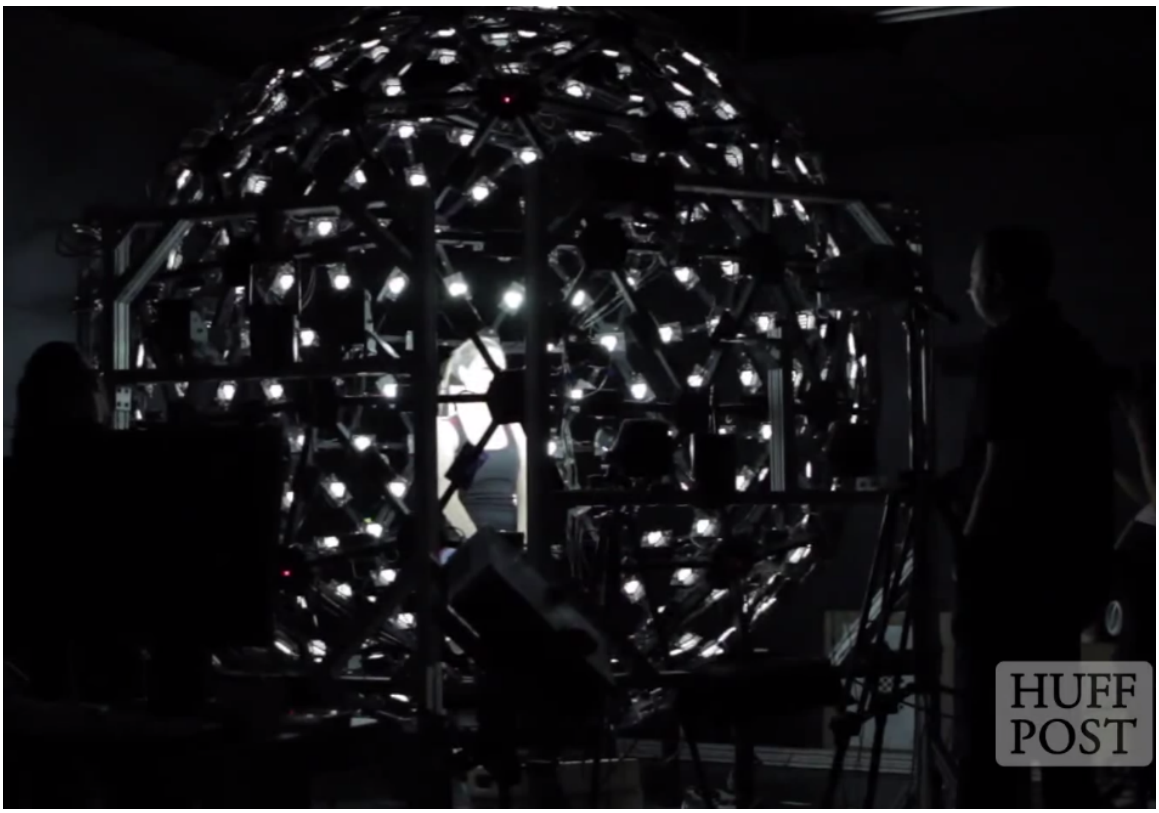


Figure 2.19: HuffPost Science correspondent Santa Maria visits Light stage X [33]

3 Reflectance acquisition setups with DSLR and Speedlite flashes

In this chapter we will discuss one of the setups that allows photometric stereo experiment to be performed. The entirety of the project will consist of two different setups including controllable LED light sources setups (as per light stages concept) discussed in Chapter 5 and a setup involving Canon camera with wireless controlled Speedlite flashes.

Henceforward in this chapter we will discuss the later. We will briefly mention how we built the infrastructure to take four different images under four different lighting conditions applicable for photometric stereo experiment. This will be done by using the Canon Camera 650D and four speedlite 430EX II flash lights, we will observe how the flash light gets controlled by the camera and discuss the underlying protocol used in doing so wirelessly, the synchronization issues will also be discussed thoroughly. In addition we will present an alternative way of controlling the flashes with camera and make comparisons on the speed and ease of use.

3.1 Setup

Firstly as part of setup implementation, we will mention the use of Canon EOS 650D series and Canon 430EXII electronic flash speedlite devices. As per the design, DSLR camera in connection to the host PC (via USB) was placed in fixed position while being controlled by a C++ program. The program (C++) was written in visual studio 2013 which on execution provided an ability to take picture, download images, control flash lights while also providing synchronization.

As shown in Figure 3.1, camera was placed in a tripod with a flash lights placed at a desired angle with respect to the object of interest. A second tripod was also used to hold the object under experiment while the flash lights are mounted to the wall in a way that acts as a directional source of light. Here we can also visualize the object of interest is placed at approximately 2 meter distance.

Similarly the background is completely covered up with black cloth so as to prevent reflection through white walls. The flashes are to be kept at similar distance with respect to the object.

Figure 3.1: Self built Reflectance acquisition setups with DSLR and Speedlite flashes

We will now briefly define each resources used during the setup and their underlying implementation.

3.2 Canon Camera 650D

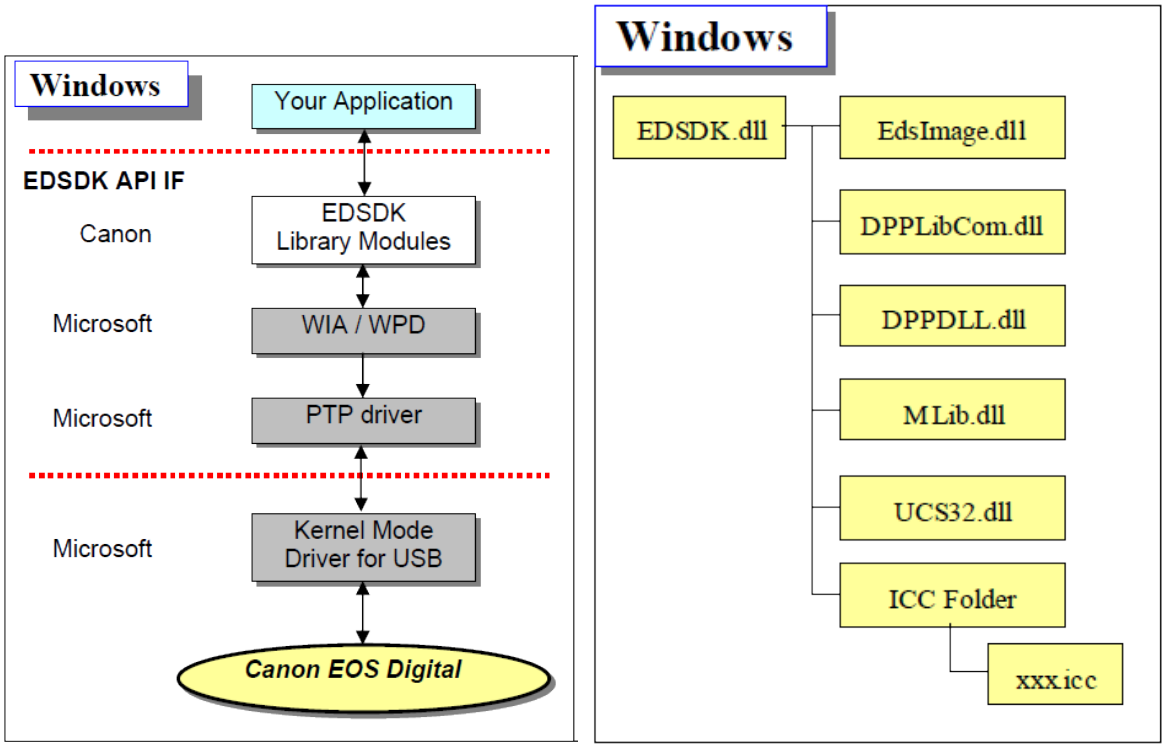
The initial approach in building programmable reflectance acquisition setups using DSLR (Digital Single-Lens Reflex) required an understanding of functionalities each resources used had to offer. The first approach was therefore to use and get familiar with the DSLR camera along with understanding the underlying implementation of the product. This led to realisation of EDSDK (EOS Digital Camera Software Development Kit) provided by Canon.

EDSDK is a development kit that provides functions required to control camera and its overall functionality. It gives an interface to access image data shot using the DSLR camera along with capability to transfer images from camera to host PC. It also offers images in RAW format e.g. .CR2 format which can be processed to obtain various mapping data that can be useful in orientation realisation. Similarly it also allows flash controls which becomes key in wirelessly controlling flash lights during image capture. The important feature it holds is providing control over the camera from a host computer. Hence this became the basic building block for the project.

3.2.1 Programming the Canon Camera 650D

Programming the Canon camera 650D was done using C++ in visual studio 2013 by using EDSDK API. The SDK followed the pipeline as shown in Figure 3.2(a) below which required an application to interact with EDSDK library modules via imports, which in return used windows WIA or WPD to retrieve images. The Picture Transfer Protocol (PTP) driver then enabled PTP devices to support the WIA driver model. USB connection was established in obtaining the required controls for camera which required Kernel mode driver for USB.

Similarly, it was important to understand the module configuration of EDSDK shown in figure 3.2 (b) in order to add on the dependencies while setting up the project in visual studio 2013.



(a) API implementation chart (b) Library Module configuration

Figure 3.2: EDSDK dependencies layout[17]

With the dependencies set for the canon camera in visual studio 2013, EDSDK objects were observed to write up a code that would control the camera from host PC. EDSDK objects employed hierarchical structure with camera list at root allowing an access to camera as shown in Figure 3.3. The hierarchy consisted of elements such as camera list, cameras, volumes, folders and image files. These elements belonged to either of the four object categories

- EdsCameraListRef
- EdsCameraRef
- EdsVolumeRef
- EdsDirectoryItemRef

This hierarchy thus allowed to create C++ wrapper around these objects which in turn could be used to control camera features from host PC. In addition to these object categories we required different object reference that would allow to control

the input and output i.e. transferring images from the camera to the host. This was possible with an introduction **EdsImageRef** and **EdsStreamRef** objects. EdsStreamRef object represented the file I/O stream which when given the download destination along with the image reference in camera, would allow downloading files to the host PC.

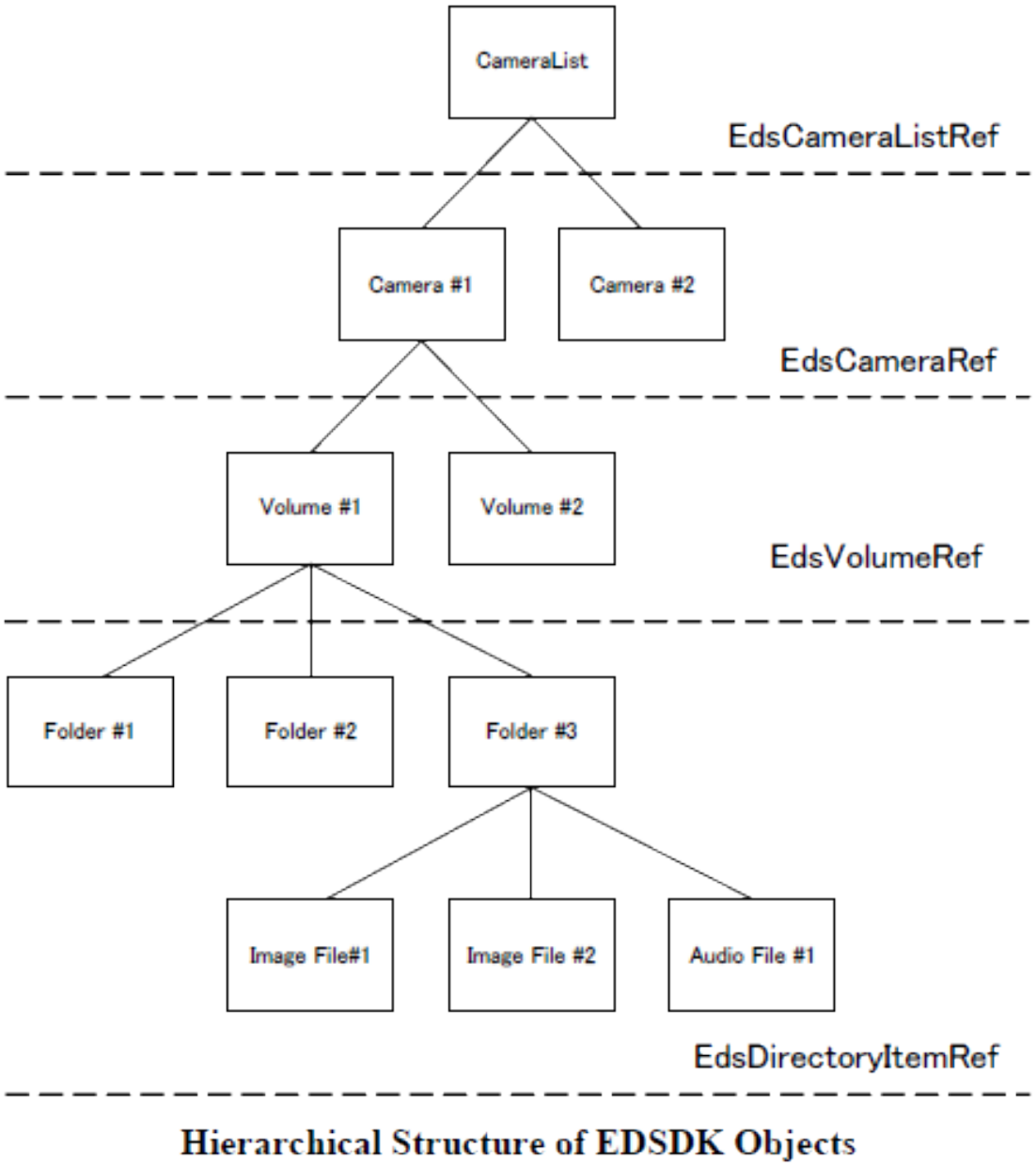


Figure 3.3

In order to make use of the EDSDK, the project had to explicitly initialize the library. For this it required to include headers files as shown in figure 8. The initialisation of SDK was done at the very beginning of the application. Alongside the initialisation, we obtain the cameras by using the *getFirstCamera (& camera)* function implemented using the above hierarchy information. If two or more cameras are connected to the USB we return the first camera detected as an object as shown in the program Figure 3.5.

```
#include "EDSDK.h"
#include "EDSDKErrors.h"
#include "EDSDKTypes.h"
```

Figure 3.4: Including necessary header files

```
camera = NULL;
isSDKLoaded = false;

// Initialize SDK
EdsError initialiseErr = EdsInitializeSDK();

if (initialiseErr == EDS_ERR_OK)
{
    cout << endl << "########################################" << endl << endl;
    cout << endl << "Initialized SDK..........." << endl;
    isSDKLoaded = true;
}

// Get first camera
if (initialiseErr == EDS_ERR_OK)
{
    initialiseErr = getFirstCamera(&camera);
    if (initialiseErr == EDS_ERR_OK)
    {
        cout << "Got Camera..........." << endl;
        this->camera = camera;
    }
    else {
        cout << "Camera Device Not Found...........";
    }
}
```

Figure 3.5: Initialising Canon SDK and retrieving Camera object

Similarly, when SDK is initialised and camera object is obtained without any errors i.e. *initialiseErr* returns 0, we require to set object, property and camera

state event handlers in order to operate routines asynchronously and handle inputs received into a program. The initialisation can be observed below in the code where *handleObjectEvent*, *handlePropertyEvent* and *handleStateEvent* are functions as parameters which are static in nature and are defined globally which takes care of the task to be performed during events.

```
// Set event handler
if (initialiseErr == EDS_ERR_OK)
{
    initialiseErr = EdsSetObjectEventHandler(camera, kEdsObjectEvent_All,
        handleObjectEvent, NULL);
}
// Set event handler
if (initialiseErr == EDS_ERR_OK)
{
    cout << "Object Event Handler Set..........." << endl;
    initialiseErr = EdsSetPropertyEventHandler(camera, kEdsPropertyEvent_All,
        handlePropertyEvent, NULL);
}
// Set event handler
if (initialiseErr == EDS_ERR_OK)
{
    cout << "Property Event Handler Set..........." << endl;
    initialiseErr = EdsSetCameraStateEventHandler(camera, kEdsStateEvent_All,
        handleStateEvent, NULL);
}
```

Figure 3.6: Setting event handlers

After initialising SDK and performing camera checks along with setting handlers, a wrapper to the EDSDK was implemented. The wrapper consisted of basic function that would allow taking pictures from the host PC, view the live screen, obtain flash information and download the image in various formats (JPEG, .CR2) as shown in the header file (Figure 3.7) below.

Furthermore, this setup with basic implementation allowed us to model a smaller version of light stage with an ability to take pictures , change settings for the camera and also download images. In addition, images processing became possible with obtained images allowing us to create applications based on image processing.

The following header file highlights the functionality the API provides

```
private:
    EdsCameraRef camera; // stores camera object obtained from getCamera function
    EdsError initialiseErr = EDS_ERR_OK;
    bool isSDKLoaded;

    EdsError downloadImage(EdsDirectoryItemRef directoryItem); // Helper function for downloadLastImage
    EdsError getFirstCamera(EdsCameraRef *camera);

public:

    Canon650DCamera();
    ~ Canon650DCamera();
    EdsCameraRef getCamera(); // Gets the installed camera as an object
    EdsError getInitialiseErr(); // Returns an error if any task you want to perform is invalid

    EdsError takePicture(EdsCameraRef camera); // takes the picture
    bool downloadLastImage(EdsCameraRef camera); // Downloads the last image in the camera

    EdsError getTv(EdsCameraRef camera, EdsUInt32 *Tv);
    EdsError getTvDesc(EdsCameraRef camera, EdsPropertyDesc *TvDesc);
    EdsError setTv(EdsCameraRef camera, EdsUInt32 TvValue);

    EdsError startLiveview(EdsCameraRef camera);
    EdsError endLiveview(EdsCameraRef camera);

    EdsError BulbStart(EdsCameraRef camera);
    EdsError BulbStop(EdsCameraRef camera);
    EdsError downloadEvfData(EdsCameraRef camera);
```

Figure 3.7: Header file for Canon DSLR class

3.3 Canon Speedlite 430EX II

Canon Speedlite 430EX II acts as an external flash to the Canon Camera. It is based on ETTL(Evaluative flash metering with preflash reading)technology i.e. the flash acts as a slave to the cameras built in flash. It then uses the preflash information from the master. Canon Camera on the other hand has a wireless flash system that allows two or more speedlite flashes to fire. A sensor is built into the external flash which senses the light from the master flash i.e. the camera built in flash and causes a remote flash to fire.

The multiple speedlite gives us the creative control over how we provide lighting to any object. Its wireless feature capability makes it easier to mount the flashes to various locations. Hence with variants in background lighting possible, it becomes suitable to use these wireless flash as our light sources during our photometric stereo implementation.

During our photometric stereo experiment it is necessary to deal with synchronisation issues between the flash and the camera and thus understanding the type of control within the canon system becomes important. There are two types of wireless flash controls used within the canon system. Radio wireless and Optical pulsed-light wireless.

3.3.1 Optical pulsed-light wireless

This system uses the pulsed optical light technique in order to send the flash settings and the triggering pulses information from master to the slave units. It requires a line-of-sight connection between the master and the slave flashes. This therefore ranges within 12 m distance. Since the pulsed light wireless technique requires line-of-sight connection and ranges over limited distance, we felt we could have synchronisation issues during our experiment and hence chose the alternative i.e. the system using Radio wireless.

3.3.2 Radio wireless

This system uses 2.4Ghz radio frequencies to control the slave flashguns in order to send flash setting and triggering information. As it transmits radio signals and require no line-of-sight connection between master and slave, it has higher range of transmission of 30 meters.

This very advantage over optical pulsed-light wireless system makes radio wireless flexible system to use. It can be hidden behind obstacles or other object around it without having to worry about the line-of-sight connection issues.

Similarly, the system can function as either a Radio master or a Radio slave which welcomes it as being the master controlling other flash units. This provides flexibility of adding five groups of slave units with 3 flashguns in each group meaning a total of 15 slaves.

Figure 3.8: Canon Speedlite 430EX II[18]

Hence radio wireless system was chosen as the idle method while implementing photometric stereo and thus canon speedlite 430EXII was used.

3.4 Synchronisation

Although the flash chosen for the photo-metric stereo technique was idle, we ran into a problem where synchronization were to be further implemented. Our idea was to make each flash lights as point light source and thus triggering each flash differently in different times would give us four different lighting setups used in photometric stereo experiment. However the cameras built in flash acted as a master, also triggering itself while sending data to its slave flashes. We would therefore have two flashes triggered, one for master and the other for the slave at the same time. Thus in order to obtain only one light source we tend to alter the firing of photograph. This was done by varying the shutter speed while controlling the lights falling onto the photographic film. It was therefore made programmatically possible by altering the exposure time for capturing picture and the exposure time for the wireless flash as described below.

Canon Camera's EDSDK API allows us to obtain the shutter speed currently set on the camera. This is done by obtaining the properties of camera and image objects using *EdsGetPropertyData* and *EdsSetPropertyData* functions. Hence the following algorithm shows how to make use of these functions to set the shutter speed of the camera.

Algorithm 1 Changing Shutter Speed algorithm

1: Initialise canon SDK and create new instance of the SDKHandler.
2: Obtain the connected cameras to the host computer and store the required camera to *EdsCameraRef* camera object.
3: Open the session with a camera and get camera settings using *EdsGetPropertyData* with one of the parameter being *kEdsPropID_Tv*.
4: From *kEdsPropID_Tv* you can obtain the data size, access type and data type the property Id for TV returns. Using this information you can set the values of the Shutter speed by calling the *EdsSetPropertyData*. The Shutter speed value varies from Bulb to 1/8000th of seconds.

The shutter speed program implemented can be seen in Appendix A.1. Now the process of getting only one flash to trigger is done by changing the shutter speed within the range between Bulb to 1/8000th of second. Through observation it became apparent that when the master flash sends a signal to the slave, the camera triggers immediately to take pictures. Thus an immediate response between camera trigger and master flash trigger meant two observed light sources (master and slave flashes). Hence with the introduction of little delay on taking pictures would mean we were delaying the time the master takes to send signal to slave flash. Thus by the time the slave flash receives signal and triggers, our camera shutter opens. This solution was therefore implemented on all flashes for suitable results.

Similarly the slave flash had a capability of high, medium and low synchronisation settings. By altering the synchronisation level and by further testing the slaves, it became apparent that we had further flexibility in synchronisation. With few attempts it became easier to deduce the values to set on the camera where the master flash would not interfere and thus we have only one light source as required.

3.5 Channels

As discussed previously, the infrastructure we are trying to build would have four different light source controlled independently for photometric stereo experiment.

Now that we have managed to get four light sources i.e. the slave canon speedlite flash, we need to trigger them separately. This is where channels play an important role. Cannon provides four such channels: both the master unit and the slave must be set to the same channel. This allows up to four photographers, each working with different channel alongside each other without their flash units interfering.

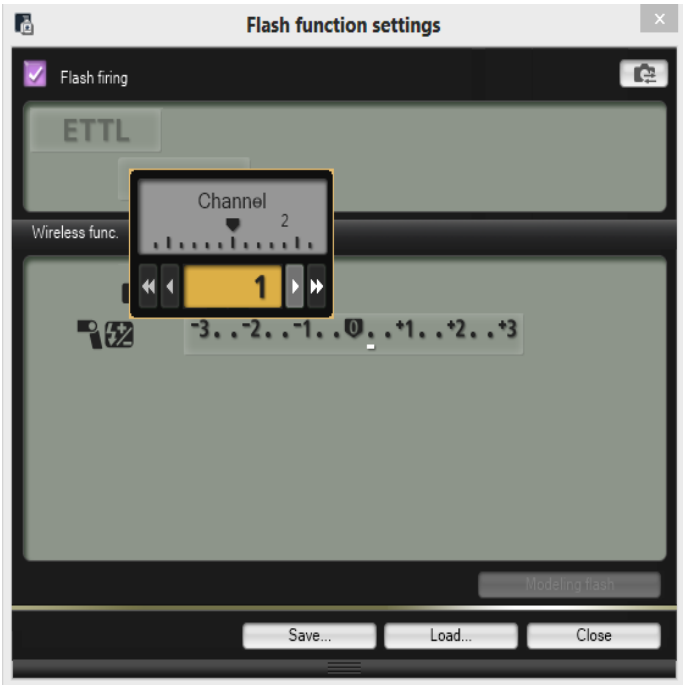


Figure 3.9: EOS utility Flash Settings[19]

Alternatively, one photographer can have four set-ups that can be controlled individually. Hence with four slaves under four different channels we can control the light sources allowing us to perform photometric stereo.

3.6 EOS utility software

An alternative approach to form the similar setup was to use the Canon's EOS utility software which provides wide range of functionality. The need for a second approach is to observe which of the two methods would bring efficiency and more control over the photometric experiment.

Once the camera is connected to the computer by a USB cable with correct setting in communication menu i.e. set to either 'Normal' or 'PC' connect, the camera controls becomes accessible. Starting EOS Utility and switching on the camera presents us with the initial screen allowing us to change camera startup settings.

With camera initialized and connected we have further access to Camera settings and Remote shooting alongside the channel update functionality as shown in Figure 3.10.

As we can see we have capability of changing channels shown in Figure 3.9 along with the capability of changing the aperture and shutter speed. Thus we will try to automatize the process of changing channels as we take pictures and also change the aperture alongside shutter speed applicable for photometric stereo experiment.

This is done by writing a C++ program that detects the EOS utility software in the windows screen. Once the position of the software is detected we monitor the mouse clicks and keyboard press. Two threads runs simultaneously with 100 millisecond on one thread responsible for changing channels. This delay allows the camera to take pictures under the channel currently set to the camera. Below is the program that shows how an update to a channel is obtained.

Figure 3.10: Canon camera setting and remote shooting[20]

```
void updateChannel(){
    INPUT inputer = { 0 };
    inputer.type = INPUT_MOUSE;

    inputer.mi.dwFlags = MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE;

    inputer.mi.dx = (long)getXPositionEOSChannel();
    inputer.mi.dy = (long)getYPositionEOSChannel();
    SendInput(1, &inputer, sizeof(INPUT));

    int j = 0;
    while (j < TIMESCHANELCHANGE){
        j++;
        inputer.type = INPUT_MOUSE;
        inputer.mi.dwFlags = (MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP);
        inputer.mi.mouseData = 0;
        inputer.mi.dwExtraInfo = NULL;
        inputer.mi.time = 0;
        SendInput(1, &inputer, sizeof(INPUT));
    }
}
```

Figure 3.11: updates channel for the canon camera

With this program up and running we write a bash script that forms a loop on executing this program four times. This process therefore allows us to take four pictures with four flashes triggered alternatively.

```
@echo off

// locating the executable which runs two threads responsible for changing channels, taking
pictures and adjusting shutter speed

for /l %%X in (1,1,4) do (   ..\..\CameraControl\Debug\CameraControl.exe

ping 127.0.0.1 -n  4)   // request a delay of 100 millisecond by pinging to localhoast
```

Figure 3.12: Bash algorithm to trigger flashes and change channels

While the camera was programmed using C++ programming language to control features in camera, an external electronic flash system was wirelessly regulated using the camera itself. With this setup as seen above, we obtained following images downloaded via C++ programming when the flash lights were placed in four different positions.

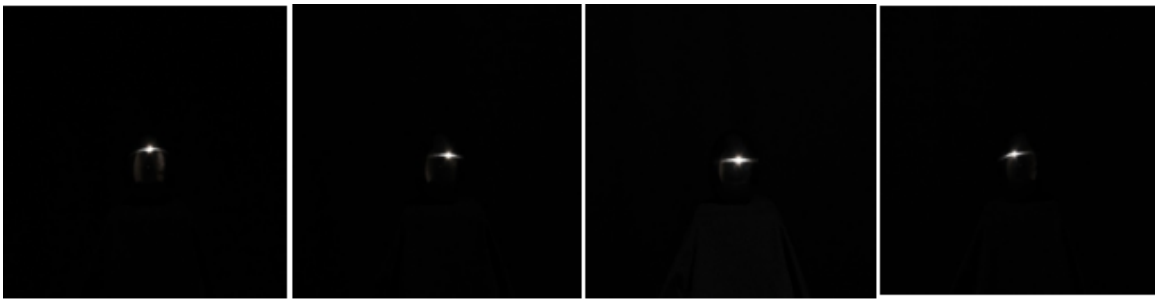


Figure 3.13: varying lighting directions

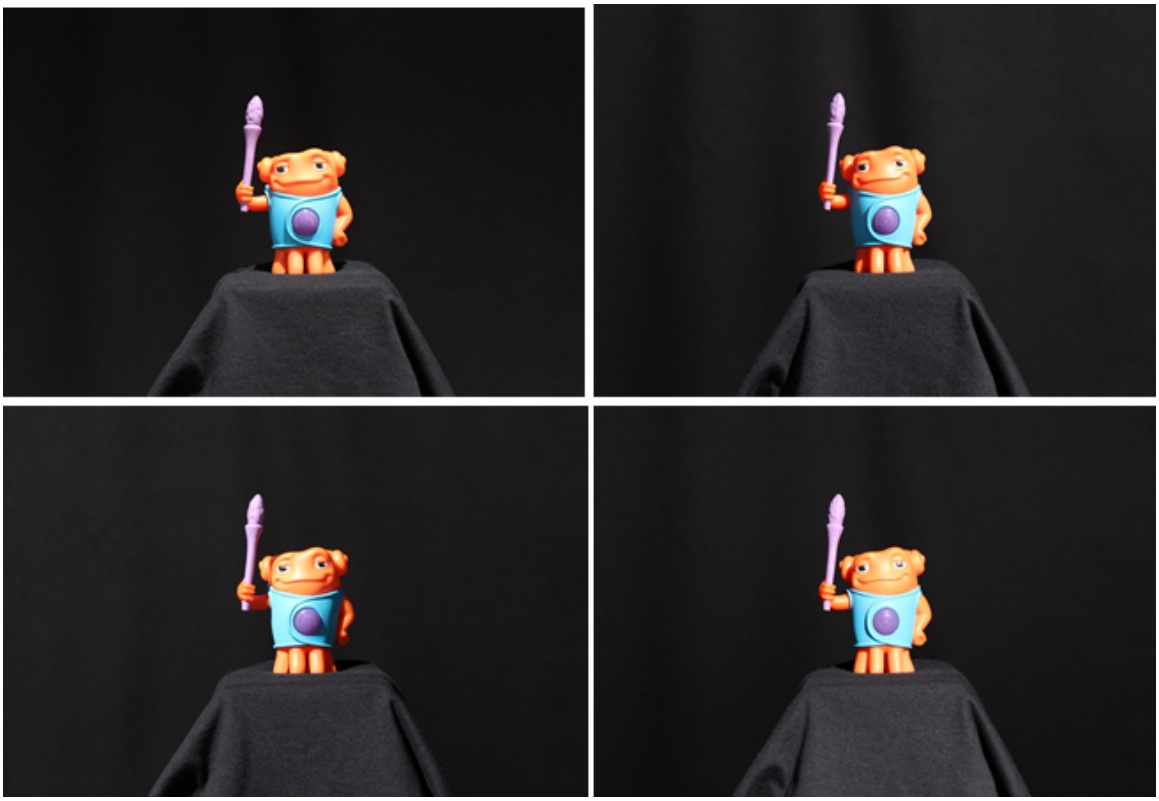


Figure 3.14: Images taken as per light direction shown above from left to right

The images seen above clearly shows how shiny the object is and how it contains specular highlights. These images therefore does not become very useful when performing photometric stereo as we require objects with Lambertian surfaces and diffuse reflection. Hence we introduce a concept of polarization discussed in next chapter.

4 | Polarization

Polarization is a property of waves which oscillates in one or more orientation. Light is an electromagnetic wave and hence exhibits polarization. Light wave is known to vibrate in a multitude of directions and in general it can be thought to vibrate in vertical or a horizontal plane. Thus a light wave that vibrates in more than one plane is called unpolarized light. The experiment followed until now had unpolarized light produced from the speedlite flash units and thus specular objects showed shininess in the resulting images.

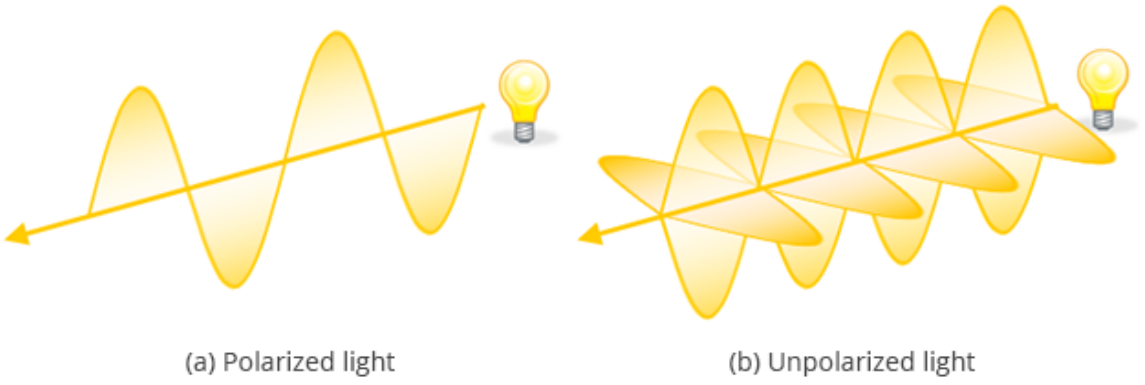


Figure 4.1: Polarization types[21]

Light emitted by the flash lights is unpolarized light because the light waves are created by electric charges, thus creating an electromagnetic wave that vibrates in a variety of directions. Hence in order to polarize the light in only one direction we make use of Polaroid filters. Polaroid filters are designed with a special material cable of blocking the electromagnetic wave vibrating at one plane while allowing the wave on other plane. In other words Polaroid acts as a device to filter one half of the vibration during the transmission of light through the filter.

Hence the output becomes one half of the light wave intensity which vibrates in single plane as shown in Figure 4.2.

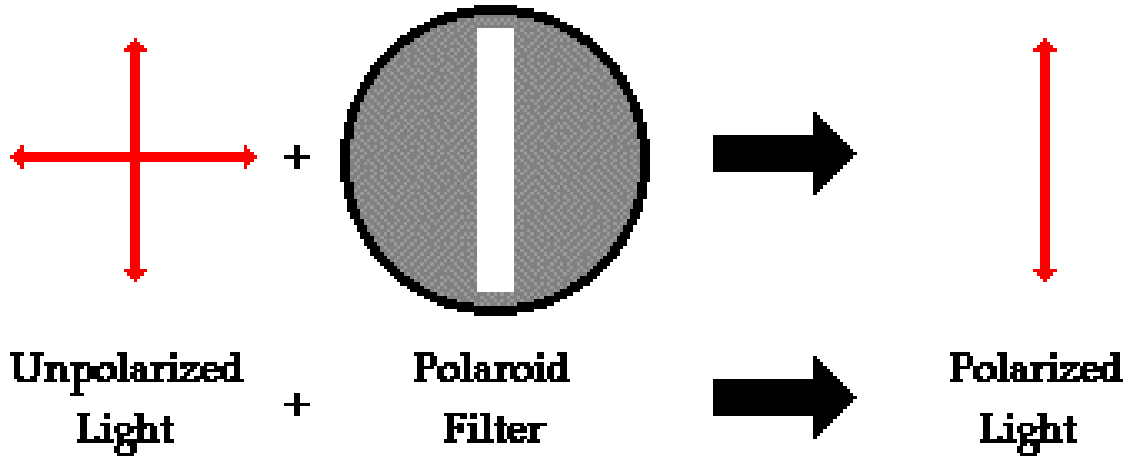


Figure 4.2: Unpolarized light to Polarized light[22]

4.1 Polarization of Reflection

As part of our experiment it also becomes necessary to understand how unpolarised light undergo polarization during reflection. The polarisation that occurs during reflection becomes directly proportional to the angle the light approaches to the surface and also the type of material it reflects upon. Metallic surface usually reflects light in many very direction and therefore becomes unpolarised. However non-metallic surfaces reflects majority of lights parallel to the reflecting surfaces plane. Thus a viewer sees a glaring effect of the object if the polarisation of light is higher.

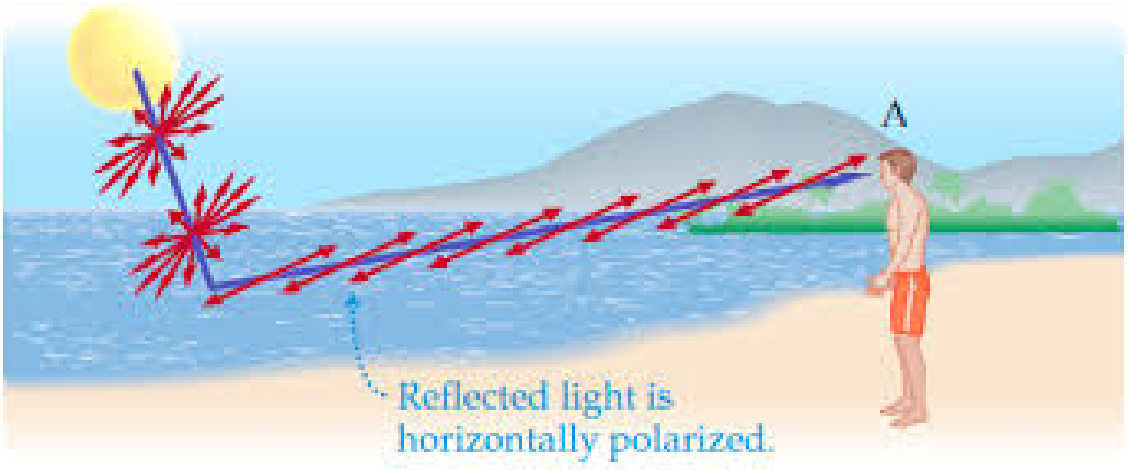


Figure 4.3: Polarization effect[23]

4.2 Cross polarization

Cross polarization is the polarization orthogonal to the polarization that is currently in place. In other words, if the light is horizontally polarized then cross-polarization would mean introduction of vertical polarizer. Hence in the experiment we conduct we use cross polarization. The camera taking the picture would be vertically polarized. This is done by mounting a vertical polarizer in front of the camera as shown in Figure 4.4.

Figure 4.4: Cross Polarized[24]

Similarly the flashes are mounted with a horizontal polarizer as shown in the figure 4.4. Now that we have cross polarized the light, we take 4 different images under this new setup and compare them with the images observed without the polarizer. We can see that the specularities seen earlier has disappeared and the resulting image is diffuse. These images now becomes applicable when performing further computations on photo-metric stereo i.e. calculating normal, albedo, Height map and getting 3D mesh of an object.

The following images shows the specular images along with the diffuse images obtained after polarization under varying lighting conditions. This therefore allowed us to visualize the differences in the result.

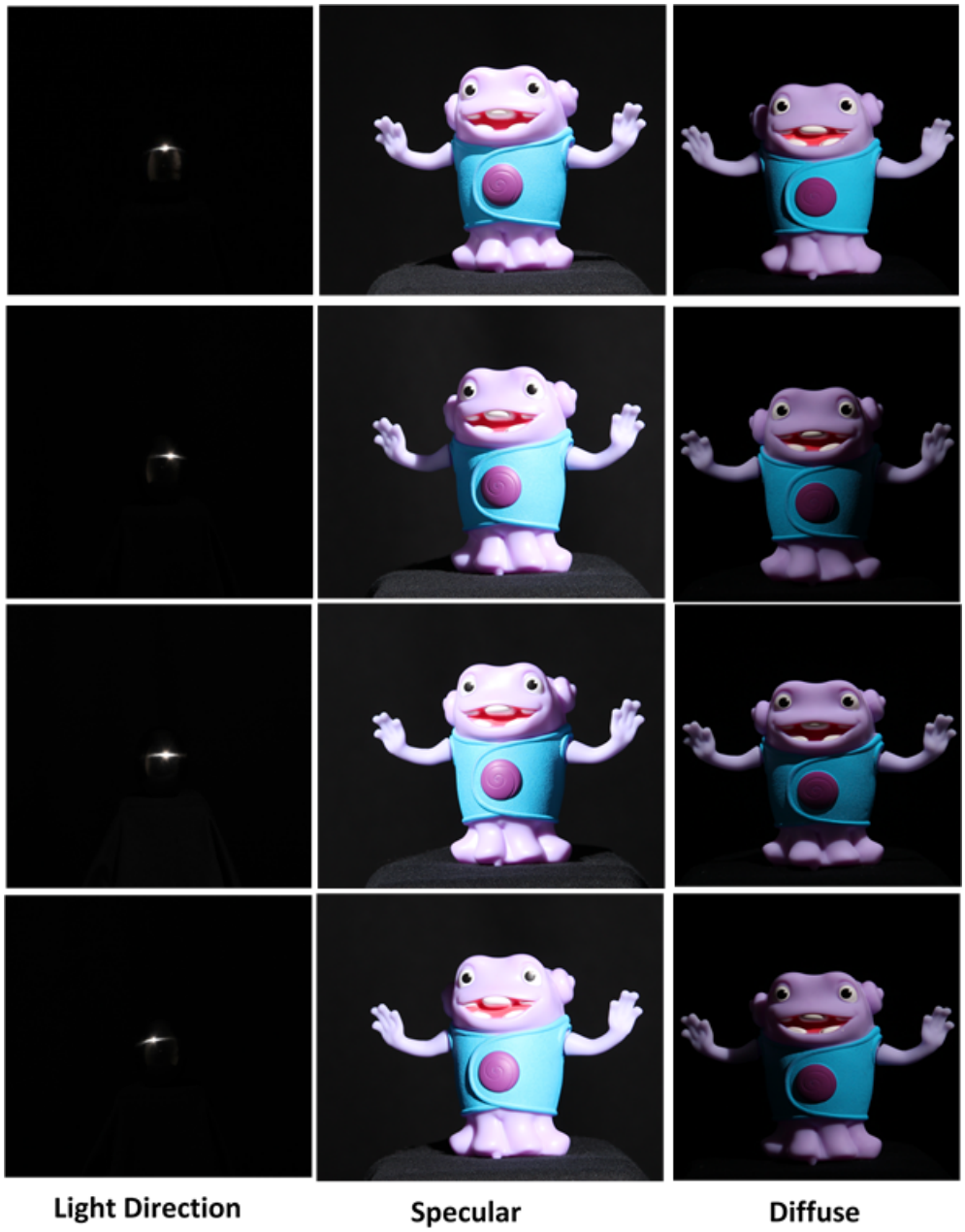


Figure 4.5: Images under polarized and unpolarized lighting conditions

As can be seen from the figure above, the specular highlights has been successfully removed after using the polarized sheet. However the object itself seem to have interreflections making the object not very useful for our experiment. The photometric experiment performed with the toy object having interreflection effects will be discussed and analyzed in Chapter 6. Let us now understand the problem of interreflection and its cause.

4.3 Interreflection

Reflectance properties of an object obtained from photo-metric stereo technique assumes that the points in an object are only illuminated by the sources of light under consideration. Thus the true reflectance value is obtained when the object has a convex surfaces. However most of the objects consists of the concave surfaces and causes the light to reflect among themselves. This very behaviour during the photo-metric stereo experiment can cause the obtained intensity values to be error prone. In return the normal map estimates along with varying reflectance (albedo) calculations becomes invalid.

4.3.1 Problem

Shree K. Nayar in his paper "Shape from Interreflections" [8] mentions how points in a scene, when illuminated, reflect light not only towards the sensor but also among themselves.

A simple interreflection example shown in figure 4.6 suggests how a light incident to the surface can have interreflections giving a glare effect. If this were to happen during the photo-metric experiment, the true intensity values for each point of an object would not be observed causing erroneous estimates throughout.

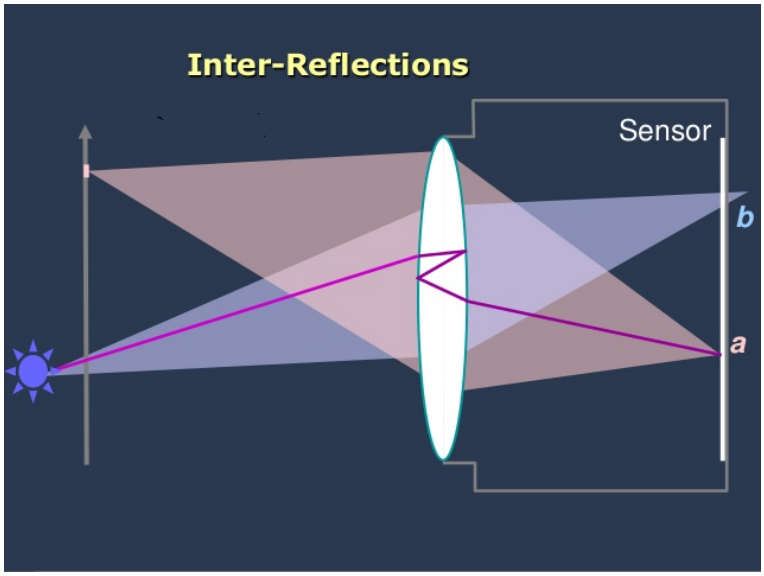


Figure 4.6: Interreflection effect[25]

These interreflections also called mutual illumination can vary the appearance of the scene. Thus if we were to choose objects of concave surfaces, the effects of interreflections can produce erroneous results. Hence the toy (Figure 3.14) we have chosen earlier during our experiment had interreflections within it and caused significant error during photometric experiment i.e. while deducing normal map, albedo and height map shown below.

Figure 4.7: Normal Map(left) and Albedo (right)

The normal map and an albedo recovered above clearly has invalid computational results as the data provided during the calculation was erroneous. Also another reason for erroneous result is because of subsurface scattering i.e. the light enters the object at one point, scatters inside it and leaves the object at another point. Thus with the help of the result above it becomes clear that the photo-metric stereo experiments becomes valid for Lambertian surfaces with diffuse reflection and no interreflection effects. Thus further experiments will consist of objects with above mentioned properties.

5 Reflectance Setup using LED Lighting Systems

At the beginning of Chapter 3 we mentioned two setups which will be of interest throughout the project. In chapter 3 we discussed one of the setup involving speedlite flash lights and camera control. This chapter will now focus on building the LED lighting infrastructure applicable for performing photometric stereo and also flexible enough to build light stages as discussed in Chapter 2. Thus a complete review on equipment's used, networking protocols followed with low level packet sniffing and reverse engineering performed to build an entire lighting network will be discussed throughout this chapter.

5.1 Setup

Figure 5.1: Self built Reflectance setup using LED lighting systems

As shown above, the design involved two PDS-70mr 24v with two Cat 5e cable connecting to the Ethernet switch. Also two data/power cable from each PDS-70mr system connects four Colour LED lamps and four white LED lamps in alternative fashion. Further connection was established using the Cat 5e Ethernet connection between the PC and the Ethernet switch forming a complete network to control the LED lamps. Note that the setup eradicated the use of Light Stage manager controller (discussed later) completely as it tend to control the lights via an API (which is entirely written by us by reverse engineering). Similarly the LED lamps were mounted on a track with a MR16 track head as seen in the complete setup (Figure 5.1).

Hereafter, a detail understanding on the underlying resources (mentioned in last paragraph) used in building the entire network with a brief explanation is shown.

5.2 PDS 70mr setup

PDS-70mr 24V is a power / data supply designed especially for MR LED lamps from Philips colour kinetics. It allows LED lighting installations with an output of maximum 72 W. It comes in three versions namely Ethernet, DMX, and Pre-programmed. We however will be using Ethernet versions as it has an Ethernet input to receive input from an Ethernet controller, such as Light System Manager and becomes flexible enough to be controlled from host PC. Figure 5.2 below shows the PDS-70mr 24V both external and internal view. It features two Ethernet port and a 24V outlet port for us to connect LED lamps as shown on the right figure below.

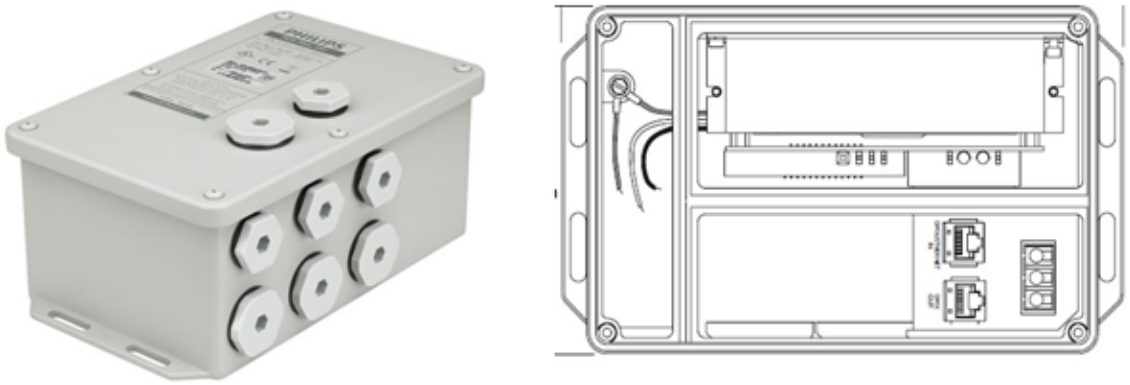


Figure 5.2: PDS 70mr[26]

Similarly we choose Ethernet version as it is not subject to the DMX addressing limitations i.e. when forming a network there is a flexibility of giving addresses to the connected LED lamps as per our necessity. Ethernet also is the preferred environment for large-scale, colour-changing light shows and video displays, both of which require large numbers of unique addresses. This therefore becomes applicable choice in long run as our project eventually desires to form a larger light stage.

In Ethernet networks, maximum data cables lengths are 328 ft. (100 m) between Ethernet devices without a repeater (for example, controller to switch, or switch to PDS-70mr 24V) and each PDS-70mr 24V device can support a maximum of 14 MR LED lamps. The fixture cable length cannot exceed 50 ft. (15.2 m). We will therefore follow aforementioned criteria to build our infrastructure.

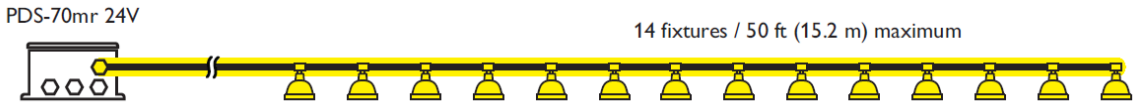


Figure 5.3: PDS 70mr connected to 14(max) fixtures[27]

5.2.1 Installation

Typical Ethernet installations use an Ethernet switch, an Ethernet controller such as Light System Manager, Ethernet Controller Keypads for push-button light show triggering, and one or more PDS-70mr 24V Ethernet devices. However here we try to avoid using Light Stage manager and instead just use host PC to control the entire network programmatically.

The installation we prefer to use is two PDS-70mr each with four alternatively placed LED lamps namely IColor MR and IW MR (discussed below in detail). We will make use of an Ethernet switch or hub to connect the controller and multiple PDS-70mr 24V Ethernet devices i.e. Each PDS-70mr 24V Ethernet device is connected to an Ethernet switch port using CAT 5e. Figure 5.4 below shows a rough design that we would wish to build for the project.

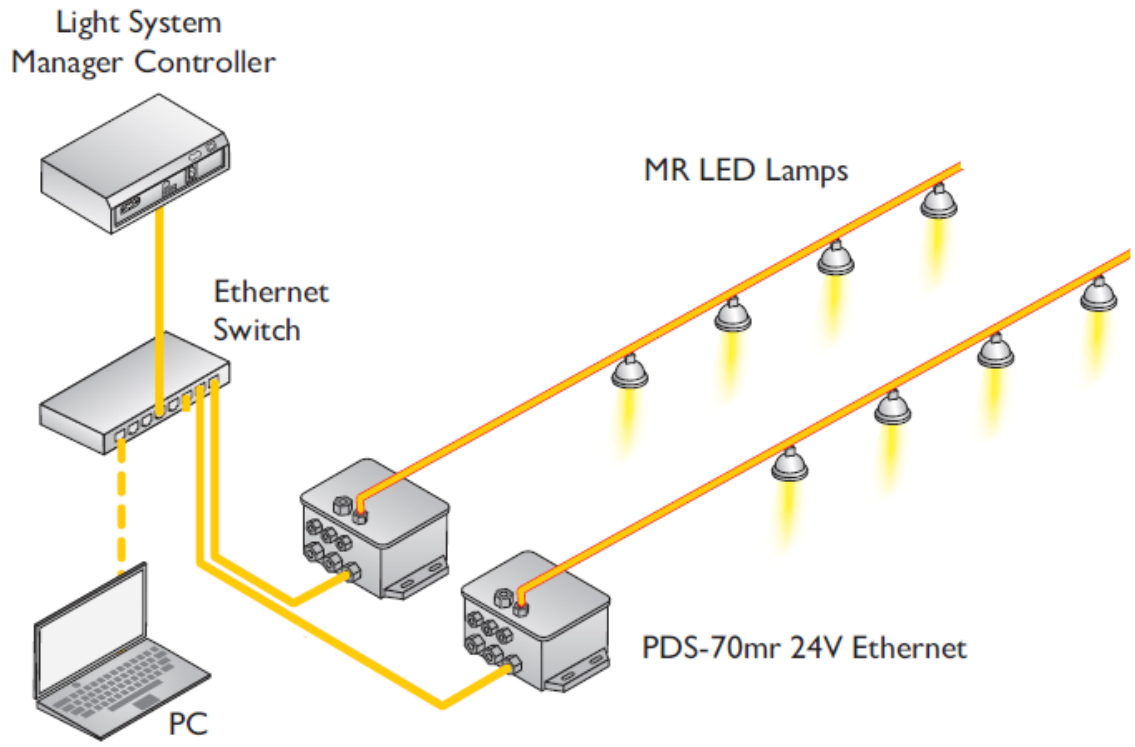


Figure 5.4: PDS 70mr installation model[27]

Similarly, the LED lamps from Philips Colour kinetics used in our setup are a unique set of lamps that has a capability of being programmed and also controlled through a network. The two set of lamps namely IColor MR Gen3 and IW MR Gen3 along with its properties and how it becomes important in photometric experiments is detailed below.

5.3 IColor MR Gen3

Firstly we will make an illustration about IColor MR gen3 and its underlying properties it holds that makes it useful for our application. The sole reason for its choice was it being an intelligent programmable colour lamp that is able to provide various colour changing effects and controllable saturated bursts of colour. It becomes useful in our photometric application as well as constructing light stages because of its high-intensity LED light sources and its capability to control each of the Red Green and Blue channels independently. However it came with three different beam angles useful in various scenarios and a choice between three different beam angles was to be made as per our need.

Figure 5.5: IColor MR Gen3 lamp[28]

The beam angle is the degree of width that light emanates from a light source. Out of the three beam angles 17, 30 and 90 degree, we made a suitable choice depending on the degree of wider spread of light and higher lumen distribution. It was therefore necessary to have further study on Polar candela distribution analysis along with Illuminance at distance and further analysis on lumens and efficacy.

The following shows a comparison between the three in terms of its polar candela distribution.

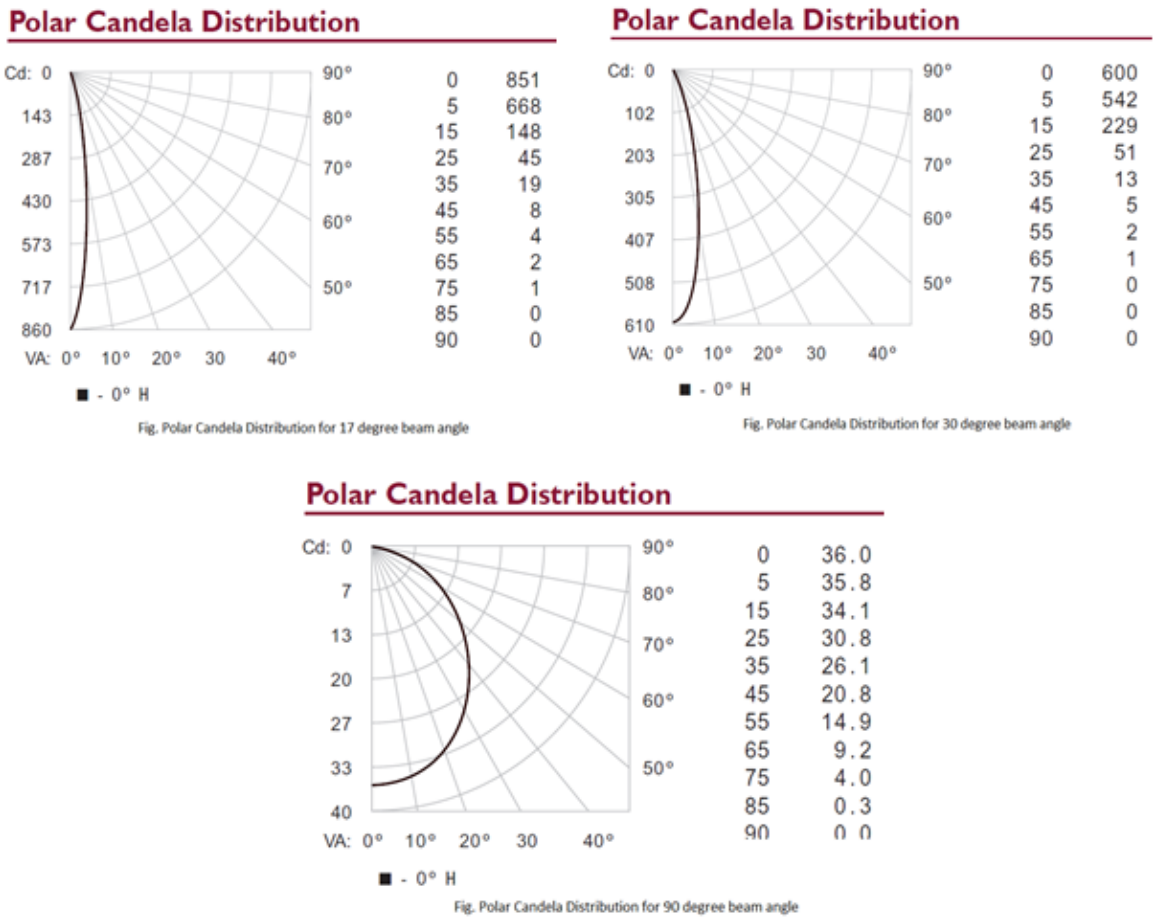
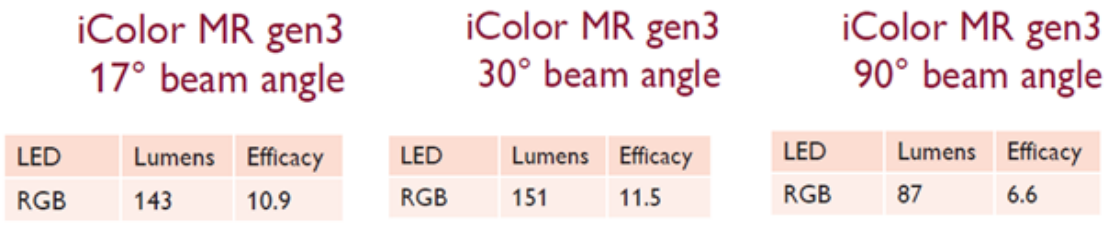


Figure 5.6: IColor MR Gen3 Polar Candela Distribution[28]

As can be seen from the Polar Candela distribution chart, 30 degree beam angle lamp provides consistent candela distribution over 0 to 25 degrees and thus it seemed applicable to have 30 degree beam angle under this analysis.

Similarly when making a comparison in terms of lumens and efficacy using the data below, it was applicable for us to have higher efficiency of 11.5 with higher lumens (151 lumens) which was given by IColor MR 30 degree beam angle. It was therefore applicable to choose a 30 degree beam angle IColor lamp.



iColor MR gen3 17° beam angle

LED	Lumens	Efficacy
RGB	143	10.9

iColor MR gen3 30° beam angle

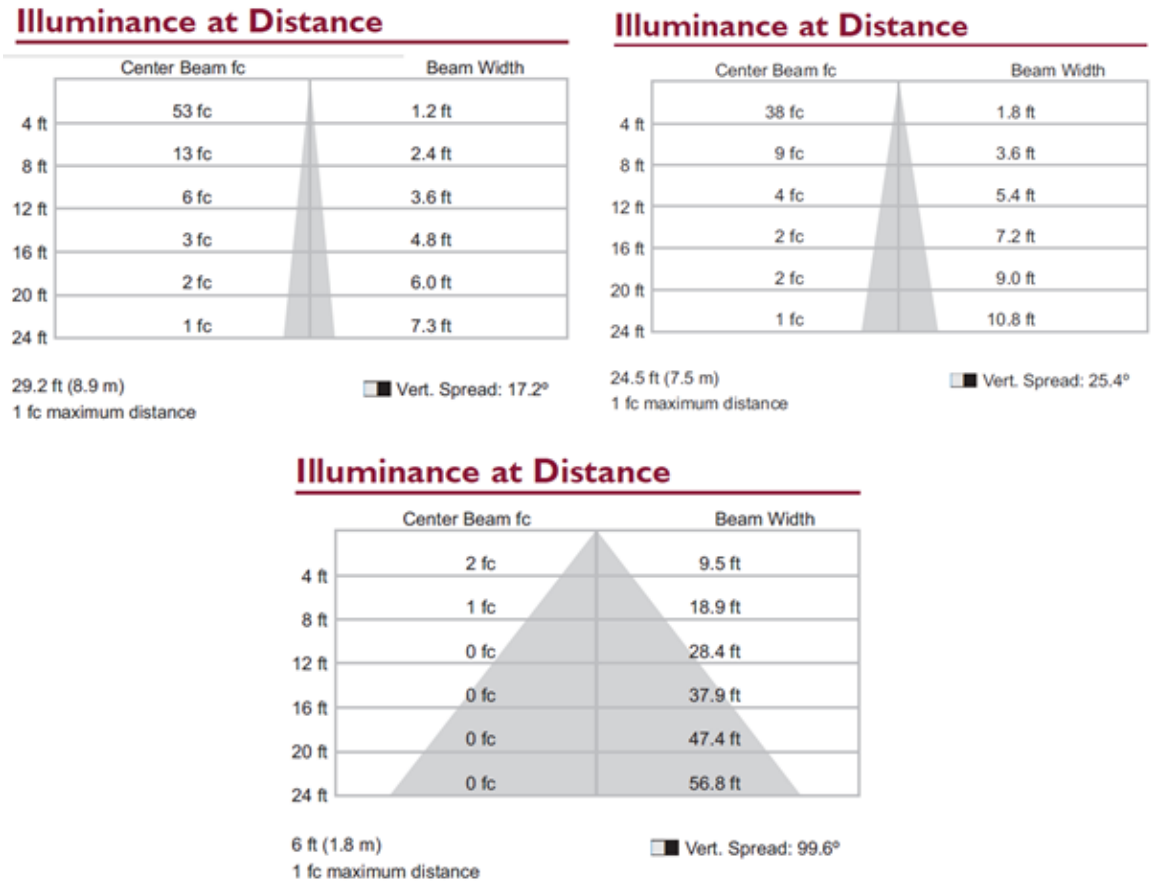
LED	Lumens	Efficacy
RGB	151	11.5

iColor MR gen3 90° beam angle

LED	Lumens	Efficacy
RGB	87	6.6

Figure 5.7: IColor MR Gen3 Lumens and Efficacy[28]

On the other hand a further analysis was done in accordance to the Illuminance at distance. The following data was used as provided by the supplier.



Illuminance at Distance

	Center Beam fc	Beam Width
4 ft	53 fc	1.2 ft
8 ft	13 fc	2.4 ft
12 ft	6 fc	3.6 ft
16 ft	3 fc	4.8 ft
20 ft	2 fc	6.0 ft
24 ft	1 fc	7.3 ft

29.2 ft (8.9 m)
1 fc maximum distance
Vert. Spread: 17.2°

Illuminance at Distance

	Center Beam fc	Beam Width
4 ft	38 fc	1.8 ft
8 ft	9 fc	3.6 ft
12 ft	4 fc	5.4 ft
16 ft	2 fc	7.2 ft
20 ft	2 fc	9.0 ft
24 ft	1 fc	10.8 ft

24.5 ft (7.5 m)
1 fc maximum distance
Vert. Spread: 25.4°

Illuminance at Distance

	Center Beam fc	Beam Width
4 ft	2 fc	9.5 ft
8 ft	1 fc	18.9 ft
12 ft	0 fc	28.4 ft
16 ft	0 fc	37.9 ft
20 ft	0 fc	47.4 ft
24 ft	0 fc	56.8 ft

6 ft (1.8 m)
1 fc maximum distance
Vert. Spread: 99.6°

Figure 5.8: IColor MR Gen3 Illuminance Distance [28]

Thus for our photometric stereo application it was applicable to have a beam width just a right amount i.e. not to low and not too high. As 30 degree beam angle IColor lamp has a beam width of 10.8 ft. and with illuminance distance of 24 ft., it was applicable for the application we chose to build. Therefore, for us to have higher degree of freedom while focusing to the object of interest during photometric stereo experiment, we chose IColor with 30 degree beam angle.

5.3.1 RGB Channels

IColor MR gen3 consists of LED channels namely Red Green and Blue channels. It has an input voltage of 24 VDC essentially provided by the PDS-70mr we discussed earlier. The power consumption is of 5W when at its maximum output at steady state. By default the lamps operate in 8-bit mode and hence channel 1 represents Red, channel 2 represents Green and channel 3 for Blue. Hence these channel operating at 8-bit mode will have 256 dimming steps. Each RGB channels can be controlled independently by giving an intensity value ranging from 0 to 255 i.e. 0 being the least bright and 255 the maximum brightness. Hence when programming the lamps it becomes necessary to discover which lamp are we considering and which channels are to be set with the intensity values.

5.4 IW MR Gen3

Similar to IColor MR Gen3 is the IW MR Gen3 lamps. IW MR Gen3 is an intelligent white LED lamp that provides intense, colour temperature modifiable white light. It provides three different channels of warm, neutral, and cool LED sources. The temperatures ranges from 2700 K - 5700 K. These high-intensity LED light sources comes in three beam angles and is suitable for our application on photometric stereo.

Figure 5.9: IW MR Gen3 Lamp[29]

As mentioned previously on the choice of beam angle for IColor MR Gen3 lamps, a similar analysis was performed to make a choice between 20, 26 and 100 degree beam angles for the white lamp. Conclusion was therefore to choose 26 degree beam angle lamp as it had higher lumens of 241 and 19.4 efficacy and would fit the purpose of the experiment.

5.4.1 Temperature Channels

IW MR Gen3 consists of three LED channels for setting the temperature of 2700K, 4000K and 5700K. Similar to IColor MR Gen3 lamp IW MR Gen3 lamps has an input voltage of 24 VDC essentially provided by the PDS-70mr we discussed earlier. The power consumption is of 5W when at its maximum output at steady state. It operates in 8-bit mode, where each fixtures use one address per LED channel (2700 K, 4000 K, and 5700 K).

Thus addressing the channels for warm, neutral and cool temperatures for each IW MR Gen3 lamps along with RGB channels for each IColor lamps becomes essential. With these channels addressed and those address capable of receiving correct intensity values via network, it becomes easier to perform experiments that requires variations of lighting effects. Therefore hereafter we discuss the underlying implementation on how we address each lamps along with how we built a network to send correct packets to those addresses.

5.5 Addressing Lamps

The lighting system from Philips Color Kinetics comprises a controller, wiring, Power / Data Supplies or Data Enablers, and fixtures. Addressing therefore enables the devices in the system to extract the correct segment of data from the data broadcast sent by the controller which is the Host PC in our case. Using the data targeted for its address, a fixture can display the correct light output. Each IColor MR Gen3 and IW MR Gen lamps has a serial number associated to it. This serial number is unique for each bulb and is looked-for while discovering the connected IColor lamps in a network.

Figure 5.10: Address of the Lamp[29]

Addressing the lamps requires a local area network built through which packets has to be sent to the specific lamp with the address you wish to assign. Since we are dealing with PDS-70mr Ethernet version and the lamps connect to it, we can configure the lamps address using a software called QuickPlay pro which will be discussed shortly. However for the experiments, we require an API which can control, discover, set intensities and assign addresses to the lamps automatically. Having said that the Philips Company (provider of the IColor and PDS-70mr) do not provide an API to perform these tasks. Hence, we will build our own API by reverse engineering the software (QuickPlay pro).

5.6 Networking

With the infrastructure built (Figure 5.1) and PDS-70mr along with Kinetic lights connected, we build a protocol through which the host PC controls the PDS-70mr along with the lamps. It required us to setup a local area network through which we send packets to the desired addresses in the network. The configured PDS-70mr Ethernet version system followed a protocol which required us to set the local area networks IP address within the range 10.1.3.00 to 10.1.3.XX where X relates to maximum two digit number. This was done by following the steps below.

- In Windows we open a Network sharing centre and choose to change settings for Ethernet network. We choose the Ethernet that we tend to use for PDS-70mr system. We then open local area connection settings as shown in Figure 5.11 below.

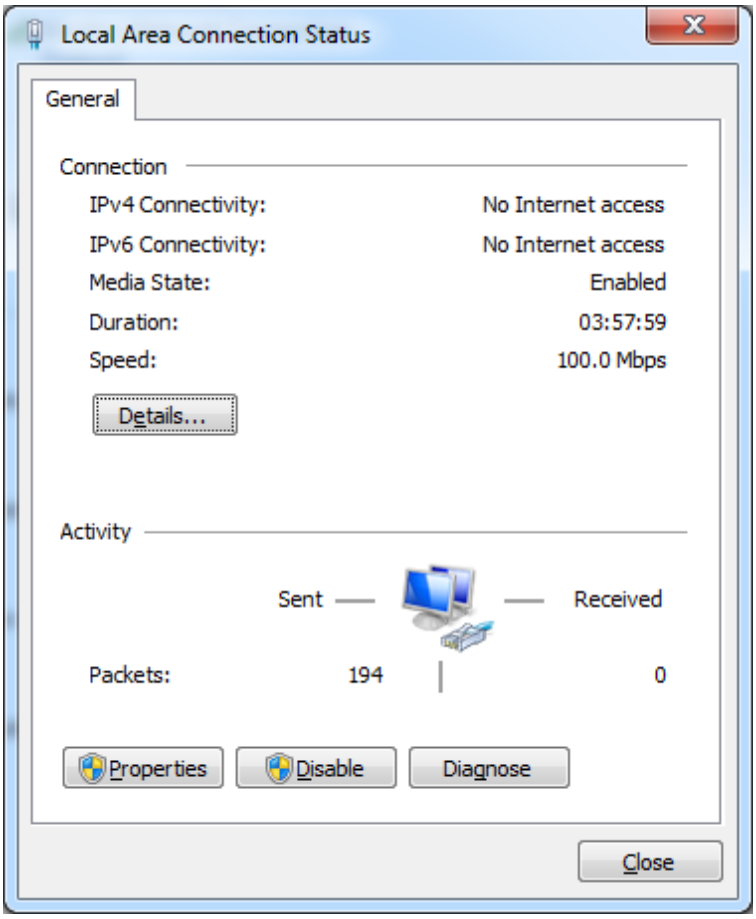


Figure 5.11: Local area network setting

- Properties tab is selected through which we select Internet Protocol version 4(TCP/IPV4) tab. This provided an option of setting a local area network under a desired IP address. As mentioned earlier the PDS-70mr is to have an IP set within the range 10.1.3.00 to 10.1.3.XX. Hence we chose to set the IP address to 10.1.3.30.

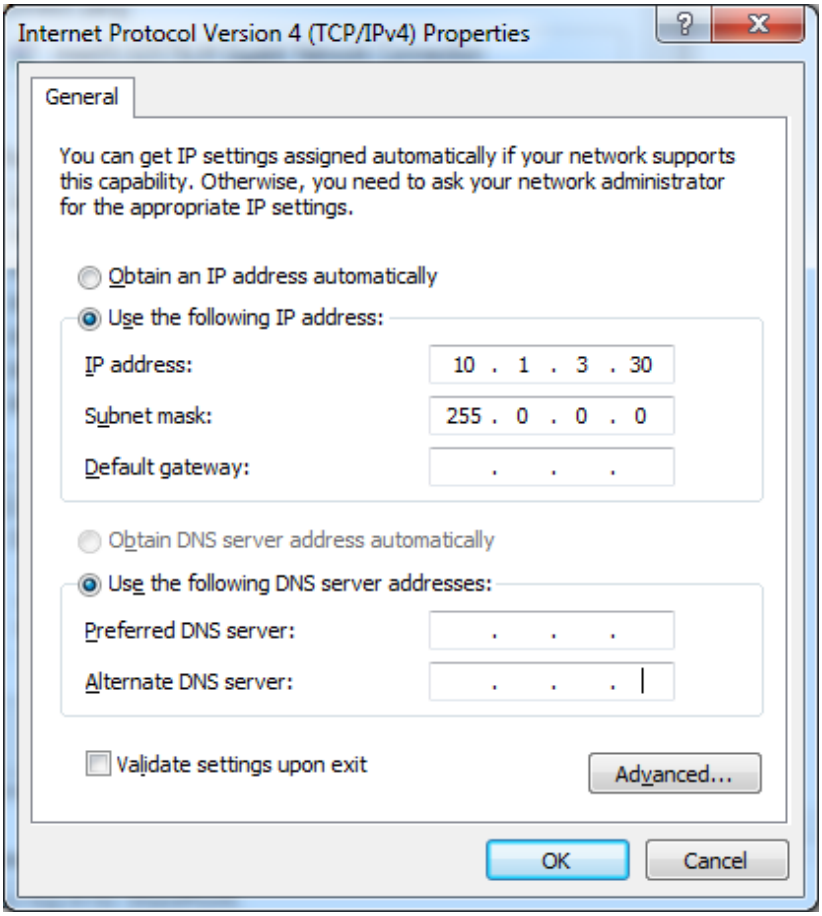


Figure 5.12: TCP/IPV4 setting

The network was now up and running and each of the resources (IColor lamps, IW MR Lamps and PDS070 MR mounted to the electric track) used were in a local area network with an IP address set to 10.1.3.30. Packets with relevant data on it was now possible to be sent through the network to control the lamps. However a choice of protocol had to be made. We had a choice of following two protocols while sending the packets to the network. Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

5.6.1 TCP or UDP

TCP is a transmission protocol that provides reliable, ordered, and error-checked delivery of data from a host to the other over an IP network. For a reliable transmission TCP establishes the connection using three way hand shaking which allows no disturbance from other sources during the entire communication i.e. it establishes connection through three way handshaking. This is done by sending and receiving packets as follows. Here a SYN and ACK (acknowledgment) packets are shared within client (cli_init_seq) and server($server\_init\_seq$).

[SYN, Seq# = cli_init_seq]

[SYN, ACK, Ack# = cli_init_seq + 1, Seq# = srv_init_seq]

[ACK, Seq# = cli_init_seq + 1, Ack# = srv_init_seq + 1]

Figure 5.13: TCP/IPV4 setting

Thus a reliable data transfer is expected. TCP also has a congestion-control mechanism that throttles the transport layer when one or more links become congested. This therefore allows reliable delivery of data. Hence no Data loss.

On the other hand UDP uses a simple connectionless transmission model with a minimum of protocol mechanism consisting of no handshaking dialogues and possess following advantages as a protocol.

- UDP has a mechanism to package data inside UDP segment and send to the network layer immediately. Hence no delay in segment transmission. Although it could face data loss, it has a capability of tolerating some data loss and try the transmission again.
- No connection establishment required for UDP thus connection can start to transfer data right away. Since setting up and tearing down a TCP connection would be wasteful.
- UDP has only 8 bytes of overhead whereas TCP has 20 bytes overhead which results in bigger packet transmission taking more time. Thus a faster response is gained.

As discussed above, the TCP is about reliability and thus extra steps such as handshaking, throttling the transport layer if congested can be time consuming. The application we tend to build and the experiment we wish to perform requires minimal time. Thus for faster and hustle free network connection and fairly reliable and quicker data transfer we chose User Datagram Protocol (UDP) as our transport protocol.

As discussed previously regarding photometric stereo experiment, we know we require four different light sources in four different locations. And it is clear that the setup we have built so far fits the purpose of the experiment. As mentioned earlier we wish to atomize the whole process of creating lighting conditions and take relevant picture i.e. we will turn on lights alternatively at a time interval and a canon camera will take picture under these lighting conditions accordingly. Thus for us to automatically control flash lights and capture images for the experiment, an API that controls these flash lights is to be built. Hence we now discuss in detail how such API was built.

Firstly we will introduce the software named QuickPlay Pro that is used to send relevant packets to the network to obtain relevant lighting effects and make a close observation on the packets. Then a C++ based API will be built by reverse engineering.

5.6.2 Quick Play Pro

Quick Play Pro is a multi-feature lighting system software from Philips that allows us to configure, test, and demonstrate lighting systems via computer. Also it allows rapidly verifying the addresses for each connected fixtures (lamps) that is configured and make changes to the addresses as per our desire. Similarly, discovering each specific channels becomes possible along with changing intensities values for each channel. The following shows the software UI which gives a rough idea on things that can be configured.

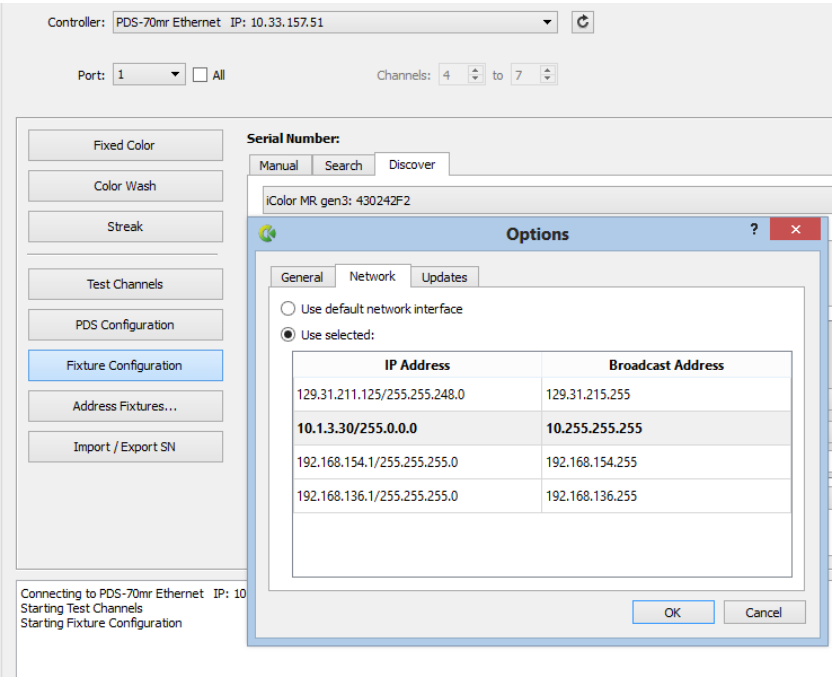


Figure 5.14: QuickplayPro settings with Tools tab[30]

With the IP address all set and Ethernet cable connected to all PDS-70 MR system, QuickPlay Pro automatically discovers Ethernet Data Enablers and Ethernet Power/ Data Supplies, and IColor/ IW MR Fixtures. As shown in the figure above we require to use the tools option to select the IP address from the list within Quick Play Pro.

As we now know that packets are constantly being sent to the network following UDP protocol. Thus in order to create a C++ based API that would allow us to perform similar task through our program, we require to understand, analyse and find patterns in the packet. Thus we introduce to a software called Wireshark that is applicable for sniffing packets in the network.

5.6.3 Reverse Engineering using Wireshark

Wireshark is an open source network packet analyser that captures network packets and displays the packet data in a detailed fashion. It identifies the live packet data from a network that you chose to analyse. Similarly it is capable of showing the hex dumps, network addresses and detailed protocol information while also capable of saving the packet data captured.

Now here we see a small example on how we obtain the desired packet information using Wireshark. Many more packets are to be examined while creating the API. However here we explain one simple example in detail for us to understand the underlying principle. The example will show how we sniff packets that are responsible for discovering the serial number of all the lights that are connected in the network for a particular PDS-70 MR system.

Firstly we open QuickPlay pro software and choose the relevant PDS-70 MR system (as we use two PDS-70 MR system in our project). Then open Wireshark software as shown below and select the Ethernet network option. The Interface setting has to be edited with an IP address of 10.1.3.30 as discussed earlier in networking section.

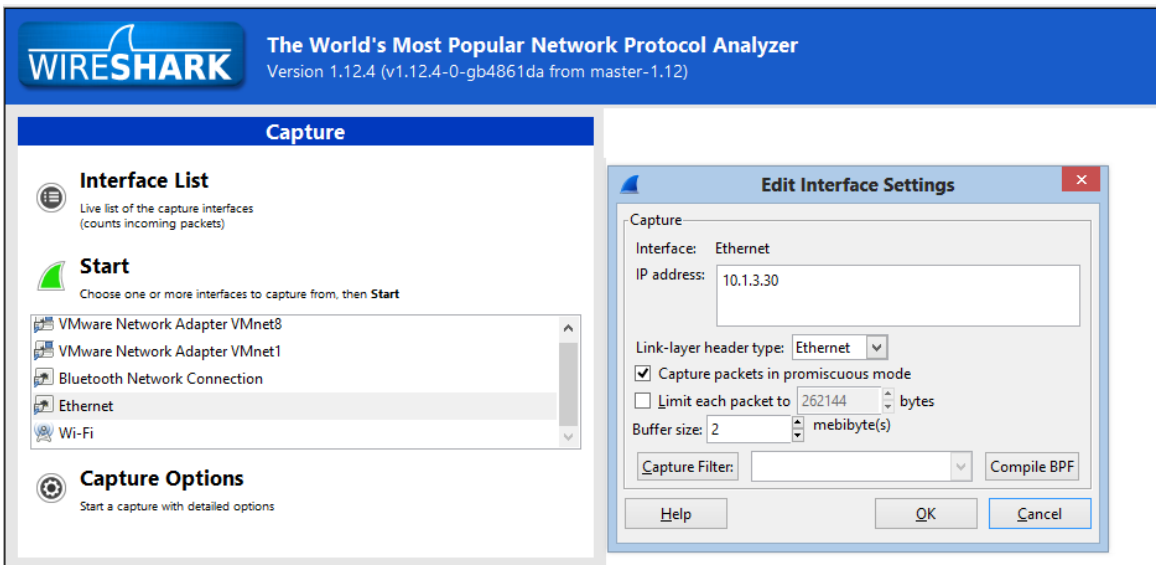


Figure 5.15: Wireshark interface setting mode[31]

With Wireshark interface setting edited and ready for reading the packets going through this interface, we try to send data packets using quick play pro. Here in this example we are trying to discover the serial number of all lights connected to the network, thus using a discover button in quick play pro we send relevant packet to the network that enables the PDS-70 MR to send back packets containing serial number information. Let us see how it looks like.

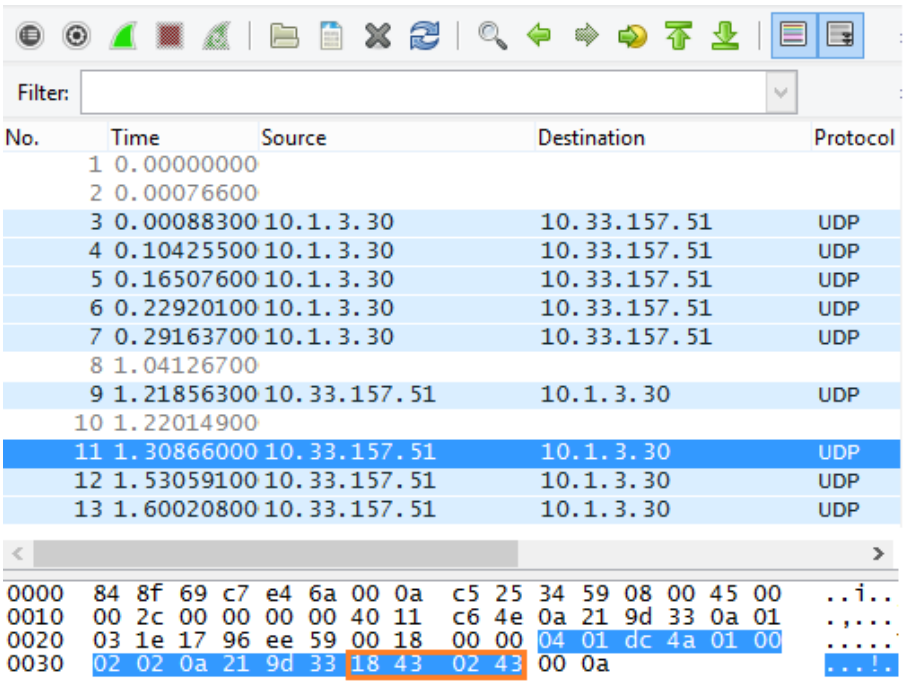


Figure 5.16: Details of the captured packets[31]

Here the first five packets are sent from an IP address set by us as 10.1.3.30 to a destination address i.e. 10.33.157.51 for the PDS-70 MR system. The first five packets (No. 3, 4, 5, 6, 7) seen in the Figure 5.16 above are UDP packets as mentioned earlier. These packets when received by the PDS-70 MR system, understands the packet as a discover packet and sends back other data packets back to 10.1.3.30 address as shown. Also in the highlighted part of the figure above we try and observe the nature of the packet. The bottom blue highlighted packet is the actual data packet in bytes. The orange highlight when reversed reads as "43024318" which is the serial number for one of the lamp connected to this system. As we can see we have received four such packets because we currently have four lamps connected to the system and each of those packet contains serial numbers of the lamps.

Thus we analyze these five sent packets from our PC and use it to discover the serial numbers for all lights connected to the network. It therefore becomes necessary to understand how each packets are read by the PDS-70 MR system and how it uses each byte in the packet to understand what lighting effects to generate. Also it becomes useful to understand what each byte represents in the protocol PDS-70 MR system follows. Thus we take one of the packets obtained from Wireshark software and deduce a method to create such packets automatically through our C++ API (will be discussed in detail later).

5.6.3.1 Packet Details

As deduced so far, 512 byte of packet is to be sent to the network where each byte associates to some task we prefer to perform. Thus we take an example of a packet and detail out what each byte represents. Below is sample packet that can be used to turn off all the lamps connected to the PDS-70 MR System. Let us now analyse the bytes for the packet.

Packet in hexadecimal with two digits signifies one byte. The below is an extraction of first 21 bytes. The use of only 21 bytes in this example is because the rest of the bytes among 512 bytes are zeros as our example is to turn off all lights in the network. Here the hex value under black box (i.e. first four bytes) represents a magic number

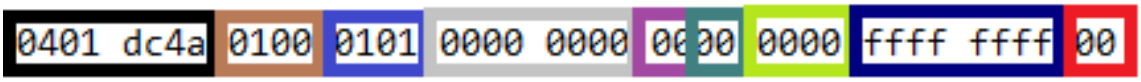


Figure 5.17: First 21 bytes of a packet

uniquely identifying the Kinetic lighting. Similarly 0100 under brown box represents a version number. The next two bytes (0101) under blue box represents the type of action. Similarly next four bytes (under grey box) is for sequence. Purple box (1 byte as 00) is for Port whereas green box (1byte as 00) is for padding. The light green box (0000), dark blue (ffff ffff) and red box (00) associates to "flags" "timer" and "unit" value respectively. "unit" here responsible for telling what operation is

currently active. For example the following are some of the operations
"00" -> Mode for setting light intensities values.
"ac" -> Mode for address discovery
"aa" -> Mode for address change
"de" -> Mode for setting intensities values depending on the serial number of lights.

The above explanation became important because the first 21 bytes for any packet is a header packet that is constant for all packets we send in a PDS-70 MR lighting system. The remaining bytes on the packet apart from these 21 bytes are responsible for either discovering lights, getting unique serial numbers for the light or even setting the intensities and temperature for the lamps in the network.

As mentioned, the type of action is represented by the light blue in the header packet above. Hence a simple discover action could be performed by changing this light blue packet of two byte (0101) to 0102. Thus a packet would be "0401 dc4a 0100 0102 0000 0000 0000 0000 ffff ffff 00" in addition to all zeroes for remaining bytes.

Although we will not go onto analysing every possible packet used while creating our C++ and Python API. Let us however see one more example of how lighting intensities can be set for a particular light in the network. There are two possible ways of telling a light to set its intensities.

- The simple but not very robust method would be to initially understand which address a light is configured to. This is done by sending discover packets and obtaining the address. If for example the address is 1. This means the 3 bytes (after the first 21 byte header packet) each represents the channels (RGB). Hence, if we would want a Red, Green and Blue channel set to their maximum intensities, we give a hex digit of "ff" for each of the three byte just after the first 21 byte as "0401 dc4a 0100 0101 0000 0000 0000 0000 ffff ffff 00 **ff ff ff** 00 00" Note the first three byte represents the lamp with address 1 set to it. Thus wishing to set intensities to lamp with address 2 would mean 4th 5th and 6th byte set as "ff" and so forth.

- However, this becomes a tedious task when the address is not as simple as 1, 2, 3 etc. Another approach would therefore be to go by the unique serial number each lamp possess. Once the discover packet receives all the packet information including its unique serial number. It now becomes possible to name the light as first or second or third or fourth light to be set to maximum intensities rather than having to calculate which byte number associates to which light. The following packet sets the intensities to maximum for a light bulb that has a serial number of **"4304005a"**

  0401 dc4a 0100 0101 0000 0000 0000 0000 ffff ffff de **4304005a** ff ff ff 0000

Here the header file has an addition of unique "unit" hex number of "de" as discussed earlier as a mode for setting intensities values depending on the serial number of lights. This is followed by the serial number of light obtained during discovery of lights in the system. And the intensities values to be set i.e. maximum intensities ("ff") for each Red, Green and Blue channels. The "ff" we have referred to so far can be altered from a range 00 to ff i.e. 0 to 255 decimal number, thus representing different intensity value.

With the above technique of identifying which packet structure performs what kind of task, it was possible to deduce the number of possible experimental task necessary for our project. Therefore a similar analysis was done with other packets and an API was created to perform following tasks.

- Detecting the network and establishing a dedicated connection between the Host PC and PDS-70 MR.
- Discovering each lamps in the network and extracting the unique serial number.
- Setting a unique address to each of the lamps discovered automatically.
- Adding a functionality to turn on lamps of choice with given intensities i.e. RGB values for IColor lamps and temperature control for IW MR Lamps.

5.7 C++ API

We hereby introduce a C++ based application programming interface (API) that is used for building a network for PDS-70MR system and Philips colour kinetic lights. It is used to interact with connected lamps in the system to control their intensities, temperatures and also deliver lighting effects accordingly. The following shows a rough sketch on the structure of the API.

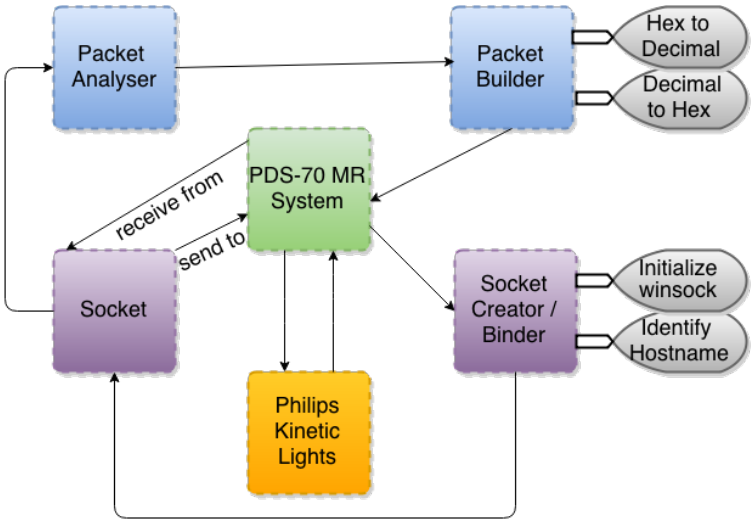


Figure 5.18: API structure

As the figure above suggests a structure where PDS-70MR system plays an important part in binding the system together. Hence the API is built based on this structure. A class named PDS-70MR is created which has the functionality as mentioned in the Figure 5.19.

```
class PDS_70mr
{
private:
    SOCKET SendSocket = INVALID_SOCKET;
    vector<string> serialNumberLights; // Gets updated when intialiseLights is called which in turn calls discover lights.
    string destinationAddress;
    unsigned short Port;

    //Helper functions
    void createSocketForConnection(); // Creates socket on automatically obtained port number for local address and binds the socket.
    void initialiseLights(); // initialises lights and also uses discoverLights function to load serialNumberLights.
    void discoverLights(); // Responsible for discovering the lights
    string PDS_70mr::getRemainingZeros(int size); // helper function to add "0" for the remaining bytes in a packets.

public:
    /*
       Takes Starting address. Usually an Integer as 1 , 2 , 3. Remember each light will need 3 channels for one light bulb.
       Takes a boolean program i.e. if you wish to program the lights initially.
       Takes the PDS_70mr destination address.. Note this has to have an IP of 10.30.1 for Kinetic lights.
       Takes the port. Also usually 6038
     */
    PDS_70mr(vector<string> serialNumberLights, int startingAddress, bool program, string destinationAddress, unsigned short Port);
    ~PDS_70mr();

    vector<PhilipsLightBulb*> Lights; // vector of object light bulbs i.e. PhilipsLightBulb class
    void PDS_70mr::turnOn(int lightnumber, int Red, int Green, int Blue);
    void turnOff();  // Overloaded function which turns of all lights
    void turnOff(int lightNumber); // Overloaded function that turns of a given light
    void programLights(int startingAddress); // programs lights starting from a given startingAddress in 3 channel based addressing.
};
```

Figure 5.19: Header file listing the functionality for PDS-70mr Class

PDS-70mr class is responsible for creating and binding sockets along with creating vector of Philips light object associated to each connected IColor or IWMR lamps. It also provides functionality on assigning address to each lamp and allows controlling them. This therefore binds all resources together and allows options to control each of these resources.

In addition, the simple code for discovering the lights is shown in the Figure 5.20.

```
void PDS_70mr::discoverLights(){
    vector<string> pkt;
    string packet = packetBuilder("discover"); // Can take any orientation of string "discover" i.e. even "DisCoVer" etc.
    for (int byteCounter = packet.size(); byteCounter < MAX_PACKET_BYTE; byteCounter++){
        packet = packet + zeroString;
    }
    pkt.push_back(packet);
    vector<string> rawPackets = sendReceivePackets(pkt);

    // Converting rawPackets to analysed packets
    analysedRecievedPacket = packetAnalyser(rawPackets);
}
```

Figure 5.20: Discovering lights in the network

Here the *discoverLights()* function gets discover packet using *packetBuilder()* function and makes the packet fill in *zeroString* ("0") to make it to the size of 512 byte. A vector of packet is formed in which the discover packet is pushed into. *sendRecievePackets()* is then used to send the packets to the network and also receive packet in *vector<string>* form if the receive flag is set. The raw packets received by the function is then analyzed using *packetAnalyser()* function which analyses the packet and extracts the serial number present in the packet. Hence all the unique serial number for the lights gets stored in the PDS_70mr system which can be used when lighting effects are to be created.

Similarly socket programming becomes the important factor in our API as it deals with sending and receiving packets throughout the network. Thus we will detail out the algorithm used in socket programming using C++.

5.7.1 Socket Programming

In socket programming a dedicated connection is established between the host PC and the system of interest (PDS-70 MR). The connection process is such that the client gets assigned a local port number and binds a socket to it. The client now can communicate with the server by writing to the socket and obtaining information from the server while reading from it.

As the platform chosen for the project was a windows platform and thus an API for windows namely Winsock was used. Winsock is a specification that defines how windows networking software should access the networking services. The following is a step by step process implemented in our programming using Winsock and socket programming techniques.

- Initialize Winsock
- Create Socket
- Obtain Host IP address and Port along with Binding Socket to it.

- Send packet through socket
- Receive packet through socket

5.7.1.1 Initialize Winsock

All processes (applications or DLLs) that call Winsock functions must initialize the use of the Windows Sockets DLL before making other Winsock functions calls. As it also makes sure that Winsock is supported on the system used, it becomes a useful initialisation process to follow. The library named ws2_32.lib consists of the necessary program files that make use of Winsock. Use of WSAStartup function loads necessary data to WSADATA variable to make initialization successful.

```
#pragma comment(lib, "ws2_32.lib")
#include <windows.h>

//----------------------
// Initialize Winsock
int wsaResult;
WSADATA wsaData;
wsaResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
if (wsaResult != NO_ERROR) {
    wprintf(L"WSAStartup failed with error: %d\n", wsaResult);
    return;
}
```

Figure 5.21: Initializing windows socket

5.7.1.2 Create Socket

After initialization, a **SOCKET** object must be instantiated for further use. This is done with the use of socket function that takes **AF_INET** (used to specify the IPv4 address family), **SOCK_DGRAM** (used to specify a data socket) and **IPPROTO_UDP** (used to specify the UDP protocol) as arguments shown below.

```
//---------------------------------------------
// Create a socket for sending data
SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (SendSocket == INVALID_SOCKET) {
    wprintf(L"socket failed with error: %ld\n", WSAGetLastError());
    WSACleanup();
    return;
}
```

Figure 5.22: Creating socket

5.7.1.3 Obtain Host IP address and Port along with Binding Socket to it

In order to accept client connection a server must be bound to a network address within the system. Thus to bind a socket we require the IP address and a port. The IP address is obtained using the *gethostname()* function which returns the host object which contains *h_addr* (IP address). The following code demonstrates how we obtain host IP address and bind a socket to it.

```
// ---------------------------------------- -
// Getting the host name and initialising the local address
// (i.e your ip address of your computer as a sender)
char host_name[128] = "";
gethostname(host_name, sizeof(host_name));
struct hostent *ent = gethostbyname(host_name);
struct in_addr ip_addr = *(struct in_addr *)(ent->h_addr);
printf("|--   Hostname: %s, was resolved to: %s\n",
    host_name, inet_ntoa(ip_addr));

sockaddr_in local;
local.sin_family = AF_INET;
local.sin_addr.s_addr = inet_addr(inet_ntoa(ip_addr));//
local.sin_port = 0; // choose any

// Binding the socket
int bindResult;
bindResult = bind(SendSocket, (SOCKADDR *)& local, sizeof (local));
if (bindResult != 0) {
    wprintf(L"bind failed with error %d\n", WSAGetLastError());
    return;
}
// ----------------------------------------
```

Figure 5.23: Getting hostname and binding sockets

5.7.1.4 Send and Receive packet through socket

The sendto and recvfrom functions are to be used to send and receive packets. Both of these functions return an integer value of the number of bytes sent or received, respectively, or an error. Each function also takes the same parameters: the active socket, a char buffer containing the packet information, number of bytes to send or receive, any flags to use and the destination address along with its size.

```
//---------------------------------------------
// Set up the RecvAddr structure with the IP address of
// the receiver and the specified port number.
sockaddr_in DestAddr;
DestAddr.sin_family = AF_INET;
DestAddr.sin_port = htons(Port);
DestAddr.sin_addr.s_addr = inet_addr(destinationAddress.c_str());

// Send a datagram to the receiver
int sendResult;
size_t pkt_length = 512;
wprintf(L"Sending a datagram to the given destination address...\n");
int j = 0;
while (j<packet.size()){
    const char * pk1 = packet[j].c_str();
    sendResult = sendto(SendSocket,
        pk1, pkt_length, 0, (SOCKADDR *)& DestAddr, sizeof (DestAddr));
    j++;
}
if (sendResult == SOCKET_ERROR) {
    wprintf(L"sendto failed with error: %d\n", WSAGetLastError());
    closesocket(SendSocket);
    WSACleanup();
    return;
}
//---------------------------------------------
```

Figure 5.24: Send packet to socket

```
//---------------------------------------------
// Call the recvfrom function to receive datagrams
// on the bound socket.
int iResult;
char RecvBuf[1024];
sockaddr_in SenderAddr = obtainSenderAddr();
int SenderAddrSize = sizeof (SenderAddr);

wprintf(L"Receiving datagrams...\n");
iResult = recvfrom(SendSocket,
            RecvBuf, BufLen, 0, (SOCKADDR *)& SenderAddr, &SenderAddrSize);

if (iResult == SOCKET_ERROR) {
    wprintf(L"recvfrom failed with error %d\n", WSAGetLastError());
}
```

Figure 5.25: Receive packet from socket

5.8 Python API

In addition to the C++ API a python API was written following the exact protocol used while writing C++ API. The features it contains are the same as that of C++ API i.e. controlling the Philips LED lighting. In addition to the packet analyzer as seen earlier which analyses the packet received, the following shows the graphical class functionality for the API. With the API developed and an understanding on

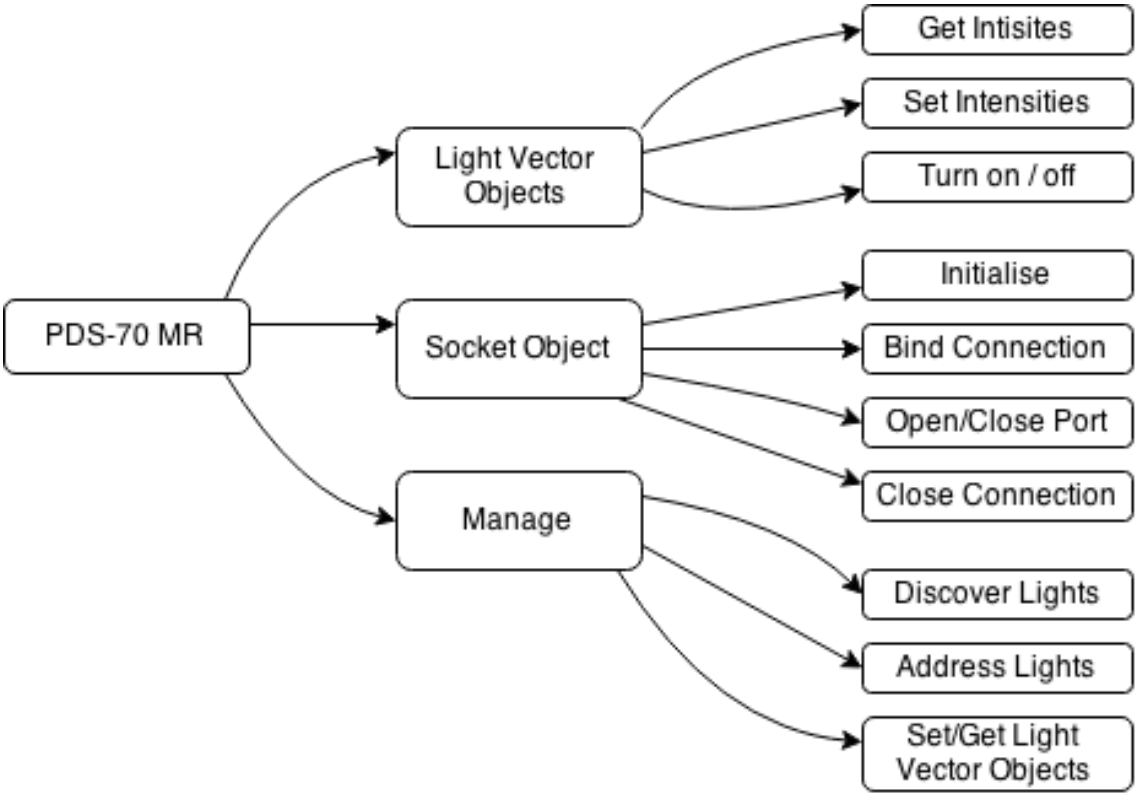


Figure 5.26: Python class functionality

how it is used is realized, the API along with the infrastructure discussed so far is put into practice to get some results for the photo-metric experiment performed in next chapter.

6 | Photometric Stereo implementation and Results

In this chapter we will perform the photometric stereo experiment using the infrastructure discussed in chapter 3 and 5. Before going on to explain and analyse results, the photometric stereo implementation in C++ alongside the use of OpenCV (APPENDIX A.2) to process images will be briefly defined. Then the raw data images obtained from the experiment will be given as an input to the C++ implementation to obtain results. The obtained result from the photometric stereo implementation in C++ will consist of the following.

- Normal Map
- Albedo
- Height Map
- 3D Mesh

The result will be analyzed and compared accordingly under all experimental setups. Let us now briefly look into the implementation of photometric stereo using C++.

6.1 Photometric Stereo Implementation in C++

As learnt previously, photometric stereo is a technique used to estimate the surface orientation and depth from images taken from a constant view but under varying lighting directions. Although the experiment becomes sufficient with only three such images, four images are taken as to minimize the noises present in the process. Also the experimental limitation it holds will be carefully managed i.e.

- Light source has to be far away from the object.
- Specular or dark spots is to be avoided as it does not output satisfactory results.
- Shadows are to be masked in order to obtain valid 3D Mesh.

An algorithm approach of implementing photometric stereo will begin with a mirror ball as an object to obtain lighting direction. The mirror ball will be placed in the experimental setup and a consecutive images will be taken under all four lighting

conditions. The mask for the mirror ball will be obtained from the image along with the light highlights on the mirror ball. The light will be calibrated to obtain the direction of light from this information. Secondly, the actual object of interest is placed in the experimental setup and images are captured in similar fashion. With these raw data in hand, we use equations to evaluate Normal map, Albedo, Height Map and obtain a 3D Mesh.

Also the following image of Buddha[32] along with the light source images will be taken as an experimental data in order to test the validity of C++ implementation. Once verified we will be using our own object to further compare and contrast the result.

Figure 6.1: Images of Buddha under given light direction[32]

C++ implementation will therefore be done in following order.

- Use OpenCV to process images and obtain a mask.
- Perform Light Calibration
- Generate Normal Map and Albedo
- Generate Height Map
- Combine the above information to get 3D Mesh.

6.1.1 OpenCV for a mask

To generate a foreground mask, a technique named Background subtraction is used. The idea is therefore to take two static image one including the object of interest and one without the object. Then a subtraction between the current frame with the object in it and the background model, produces a foreground mask as shown below.

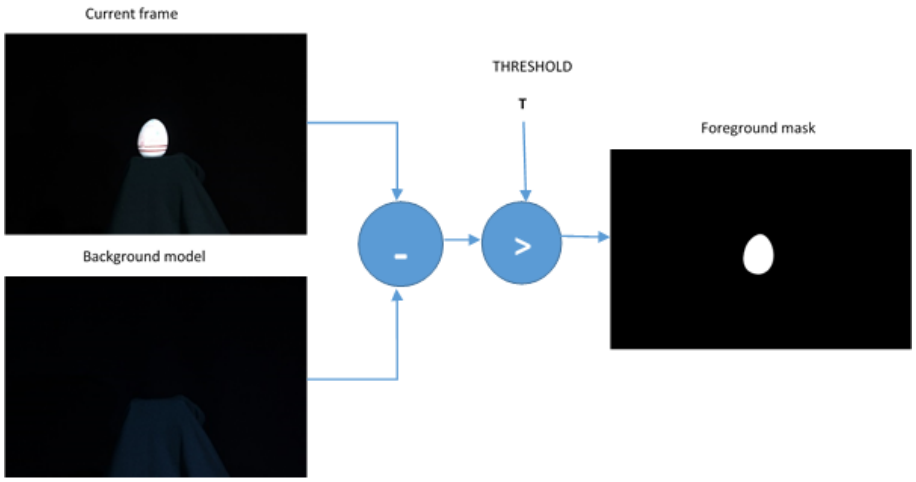


Figure 6.2: Subtraction of current frame image from background model to get mask

This can be done with OpenCV by using two such classes namely *cv::BackgroundSubtractor* and *cv::imwrite*. *cv::BackgroundSubtractor* class contains a method called *cv::createBackgroundSubtractorMOG* which creates the object that is used to generate the foreground masks. The *operator()* method in this class is performed in a loop i.e. on every frame to calculate the foreground mask and updating the background. Finally with the mask created *cv::imwrite* class allows to save it as an image. A short program shown below follows the algorithm discussed.

```
void obtainMask(char* fistFrameFilename) {
    Mat frame; //current frame
    Mat fgMaskMOG; //fg mask generated by MOG method
    Ptr<BackgroundSubtractor> pMOG; //MOG Background subtractor

    // Open first image frame which is the background model image.
    frame = imread(fistFrameFilename);
    if (!frame.data){
        exit(EXIT_FAILURE);
    }

    while ((char)keyboard != 'q' && (char)keyboard != 27){
        pMOG->operator()(frame, fgMaskMOG);
        rectangle(frame, cv::Point(10, 2), cv::Point(100, 20),
            cv::Scalar(255, 255, 255), -1);
        putText(frame, frameNumberString.c_str(), cv::Point(15, 15),
            FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));

        string nextFrameFilename = getNextFileName(frame); // gets the next filename based on the current frame details

        frame = imread(nextFrameFilename); //open Second Image
        if (!frame.data){
            exit(EXIT_FAILURE);
        }
    }
    imwrite("../images/output.jpg", fgMaskMOG);
}
```

Figure 6.3: Subtraction of current frame image from background model to get mask

6.1.2 Light Calibration

Calibrating the light source to estimate the light direction is the initial step that has to be taken when calculating the normal map. Out of many possibilities a technique involving mirror ball that when placed under a light source allows us to identify the direction of light as per its brightest spot.

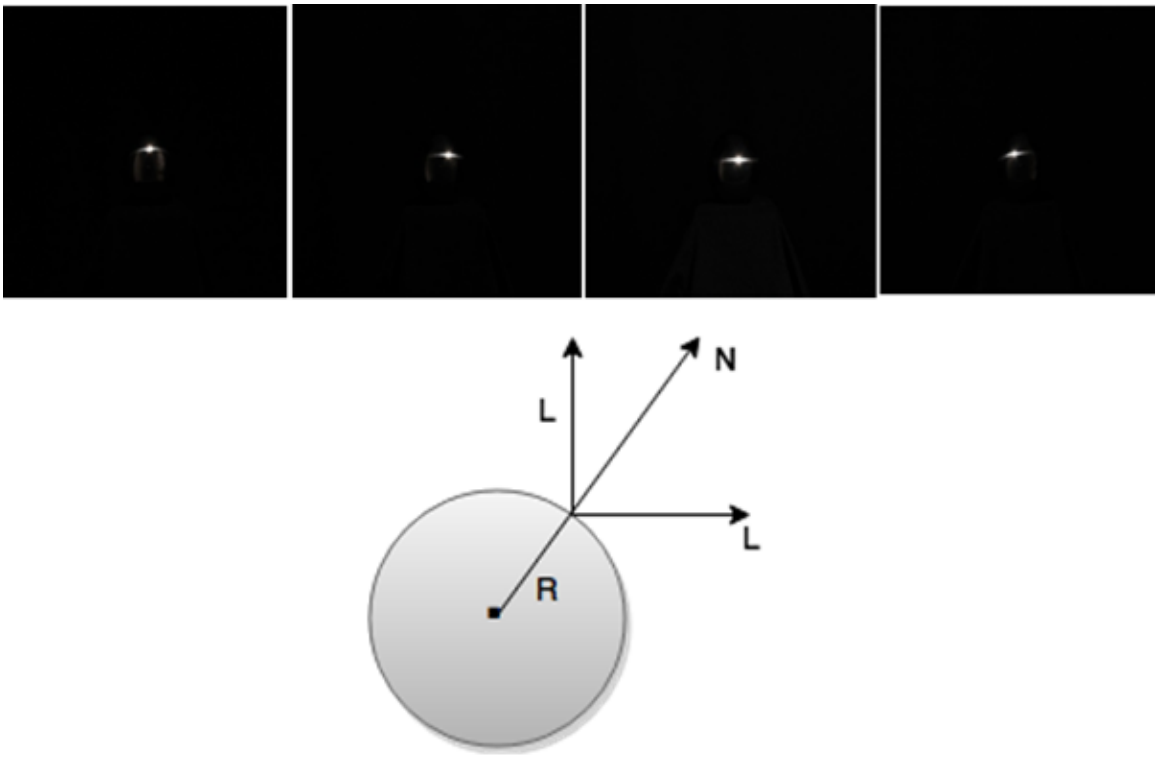


Figure 6.4: Light calibration

R here is the view direction i.e. the direction of the camera which is taken as [0, 0, 1]. Let us consider a location of the brightest point on the mirror ball as [Px , Py]. Similarly, the centre of the mirror ball as [Cx , Cy]. N here represents the Normal to the brightest point as [Nx,Ny,Nz]. The desired calculation is the light direction (L) which will be deduced using the following equation.

$$\mathbf{L} = \mathbf{2}(\mathbf{N}.\mathbf{R})\mathbf{N} - \mathbf{R}$$

where

$$\begin{aligned} N_x &= P_x - C_x \\ N_y &= P_y - C_y \\ N_z &= \sqrt{(R^2 - N_x^2 - N_y^2)} \end{aligned}$$

The centre of the mirror ball [Cx , Cy] is obtained from the mask image of the mirror ball. Firstly the image of the mirror ball is taken and a mask image is obtained using the obtainMask() function discussed earlier. The mask is again cropped in such a way that the entire circular mirror ball lies in a rectangle. To do so we use the following C++ code along with OpenCV.

```
cv::Rect LightDirection::getBoundingRectForMask(Mat Mask) {

    std::vector<std::vector<cv::Point> > outputContour;
    // finds the contour by identifying the white and black pixels
    cv::findContours(Mask.clone(), outputContour, CV_RETR_LIST, CV_CHAIN_APPROX_NONE);
    assert(outputContour.size() > 0);
    return cv::boundingRect(outputContour[0]);
}
```

Figure 6.5: Getting a mask cropped such that it lies inside a rectangle

The following is the cropped rectangle image obtained when a mask is given as an input. Thus the function outputs a rectangle containing the circular mask of the

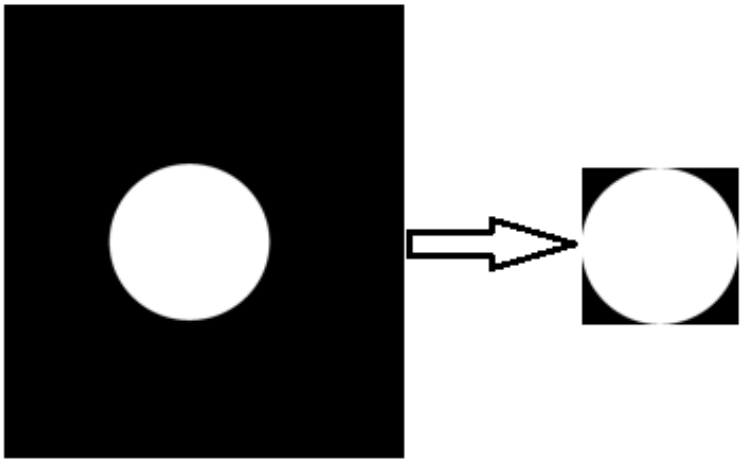


Figure 6.6: Output of using *getBoundingRectforMask* function

mirror ball. Therefore the centre of the mirror ball becomes the width and height of the rectangle. i.e. Cx is the width of the rectangle and Cy is the height of the rectangle. Hence using Cx , Cy in above equation it is possible to obtain the light direction (L).

6.1.3 Normal Map and Albedo

As discussed in Chapter 2(Background), intensity at any point on a Lambertian surface is given by
Intensity (I) = Albedo Constant (kd) * Normal (N) dot Reflected Light direction (L)
i.e. $I = K_d N.L$.....(1)

Thus to determine Normal (N) we take four different light sources and four different Intensity values to get four different equations as

$$I_1 = \kappa_d(N_x * L_{1x} + N_y * L_{1y} + N_z * L_{1z})$$
$$I_2 = \kappa_d(N_x * L_{2x} + N_y * L_{2y} + N_z * L_{2z})$$
$$I_3 = \kappa_d(N_x * L_{3x} + N_y * L_{3y} + N_z * L_{3z})$$
$$I_4 = \kappa_d(N_x * L_{4x} + N_y * L_{4y} + N_z * L_{4z})$$

To solve above equations we will rearrange equation (1) as
$L^{-1}I = k_d N$

Now using openCV we inverse the light direction calculated earlier as

```
cv::invert(Lights, lightDirection, cv::DECOMP_SVD);
```

Similarly the intensities for (each pixel) all four images can be loaded into a vector of float. Note here the *modeImageGrey* is a vector of openCV Mat that has all four images loaded into it.

```
// Vector of intensities
Vec<float, NUM_IMGSS> I;
for (int i = 0; i < NUM_IMGS; i++) {
    // Loading the grey scale image for each pixel at x,y (for all four images)
    I[i] = modelImagesGrey[i].at<uchar>(y, x);
}
```

Thus a normal can be calculated by getting the matrix multiplication between inverse light direction and intensities as shown below

```
cv::Mat normal = lightDirection * cv::Mat(I);
```

However we did not mention the albedo constant k_d in above calculation. Thus we calculate the albedo constant by dividing the normal by the square root of its dot product. Also to calculate the true normal when albedo constant is taken into account is shown below.

```
// getting the albedo here
float albedoConstant = sqrt(cv::Mat(normal).dot(normal));
if (albedoConstant > 0) {
    normal = normal / albedoConstant;
}
```

This above calculation is repeated for each pixel and a normal map is obtained. The following shows the comparison of true normal map against our calculated normal map. We observe that the result obtained from our implementation is identical to that of the original provided normal map.

Figure 6.7: Normal map for buddha obtained from original source(right) vs our implementation (left)

Similarly the identical albedo shows the C++ implementation for the provided images is valid.

Figure 6.8: Albedo for buddha obtained from original source(right) vs our implementation (left)

6.1.4 Height Map and 3D Mesh

Heightmap is a raster image that contains one channel specifying the height from the floor which means when specified with the amount of displacement, the contrast in the raster image corresponds to the height. In a heightmap, surface displacement information is often represented as black denoting the minimum height and white denoting the maximum height.

In order for us to render bumps of an object as true geometry in a 3D mesh, we require to modify the vertex height. Thus a height map is applied on the terrain to obtain such bumps on the mesh. Here the height values get mapped to a regular mesh grid with recorded value in the heightmap specifying the distance between vertices.

Furthermore the normal map obtained in previous part will be used to get the gradients in x (PGrad) and y (QGrad) axis as

```
Pgrads.at<float>(cv::Point(x, y)) = normal.at<float>(0, 0) / normal.at<float>(2, 0);
Qgrads.at<float>(cv::Point(x, y)) = normal.at<float>(1, 0) / normal.at<float>(2, 0);
```

With P and Q gradients deduced, further integration is done as discussed in Chapter (Background) to obtain the height map. Note the heightmap below (left) shows the black colour pixels are closer to the viewer while the white are further away i.e. the darkest black signifies the closest pixels to the viewer. In addition, the 3D mesh is obtained by using the integrated height map and C++ based VTK API discussed in APENDIX A.2. The following 3D Mesh (right) was recovered.

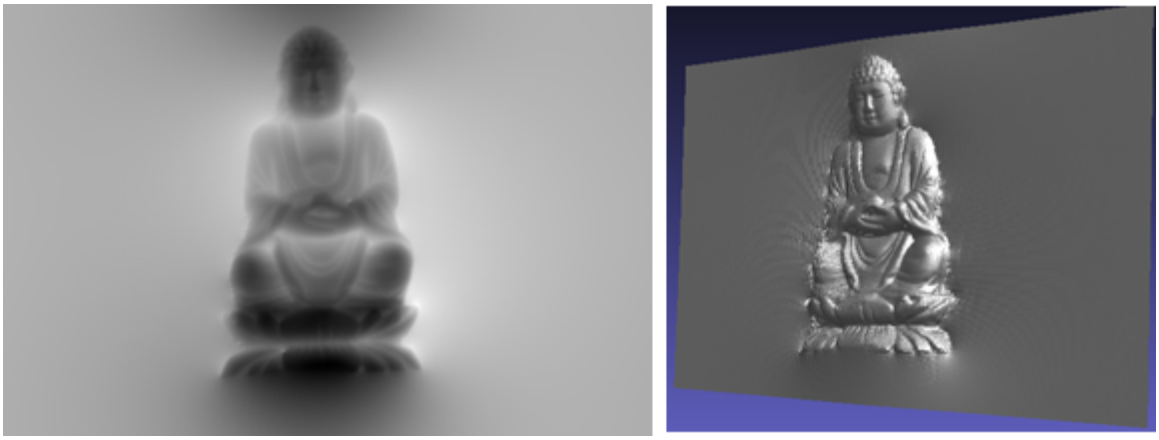


Figure 6.9: Height Map(left) and 3D Mesh(right) for buddha

6.2 Results

With infrastructures built and implementation carried out, we performed various experiments under varying setups. Hereafter we compare the results obtained from all possible setup and analyze the result. Also, the objects used throughout the experiments is shown below.

Figure 6.10: Experimental objects

6.2.1 With Canon Speedlite Flash lights

Infrastructure type : - Canon Camera with Polarised Canon Speedlite Flash Lights

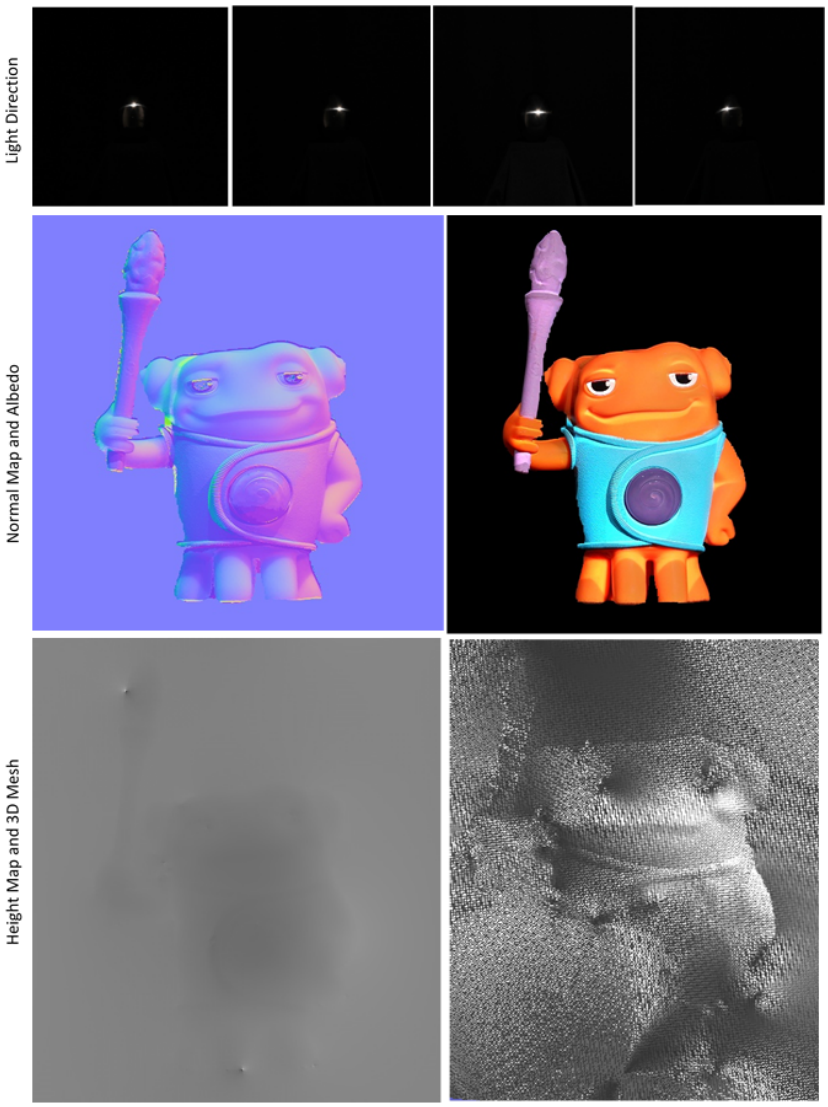


Figure 6.11: Results for Tango object under Polarised Canon Speedlite Flash Light setting

The above results from the object Tango shows the normal map, Albedo along with Height map and 3D mesh. The results however seem to have lot of noise and the images taken into consideration was not fully valid for the photometric experiment. It was understood that the reason behind the erroneous data obtained was the object under the experiment had sub surface scattering and interreflection within it as discussed in Chapter 4. Thus a further experiment with the object Egg was performed under the exact setup and lighting conditions to obtain the following result.

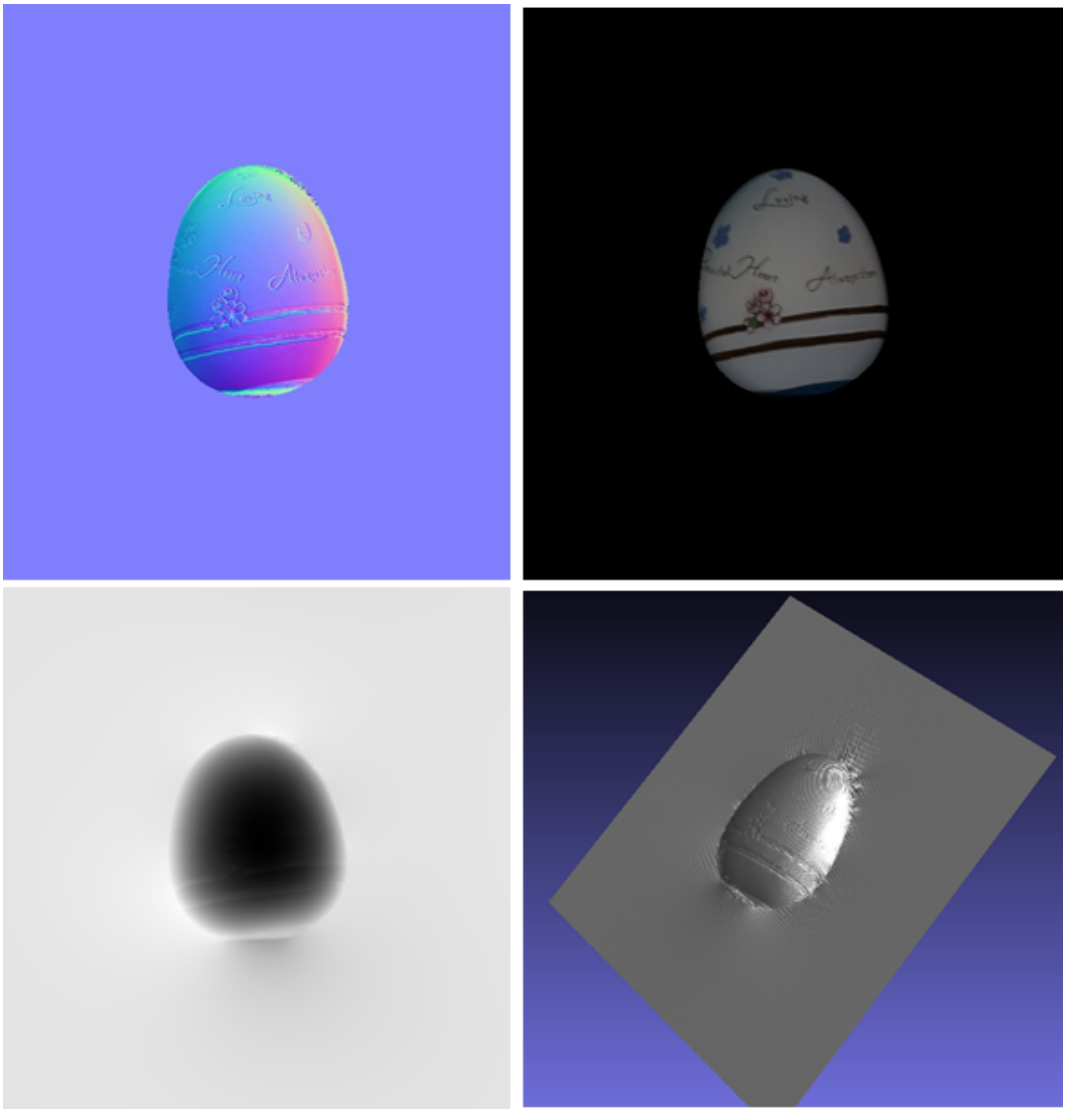


Figure 6.12: Results for Egg object under Polarised Canon Speedlite Flash Light settings

As the above result shows the Height map obtained using the Normal map and further used to get the 3D mesh is visually valid compared to that of previous object (Tango) result, it was conclusive that the setup for the experiment (Canon Camera with Speedlite flashes) was built successfully. The normal map for egg has higher detail and least of the noise compared to that of object Tango. The 3D mesh has been fully recovered and has a smoother surface detail. This therefore makes object Egg applicable for photometric stereo experiments.

Hereafter the results obtained will be based on the setup discussed in Chapter 5 with IW MR Lamps and IColour MR Lamps. Note that all of the lights under consideration are polarized lights.

6.2.2 With IColor lamps

The following result compares results from three setups. Firstly IColor lamps with its Red Green Blue channels to maximum intensities. Secondly with IColor lamps with its Red (only) channel to maximum intensity and finally with its Green channel set to maximum intensity. The object is an orange for this experiment and hence Blue(only) channel for IColor MR lamp will be obsolete as orange itself has no blue reflectance giving no response to the camera.

The light direction for the following three setups is shown below.

Figure 6.13: Light Direction for RGB lights as reflected in the mirror ball

Infrastructure type
: - Canon Camera with IColor MR Lamp
: - Full beam i.e. Max value for Red(R), Green (G) and Blue (B) channels

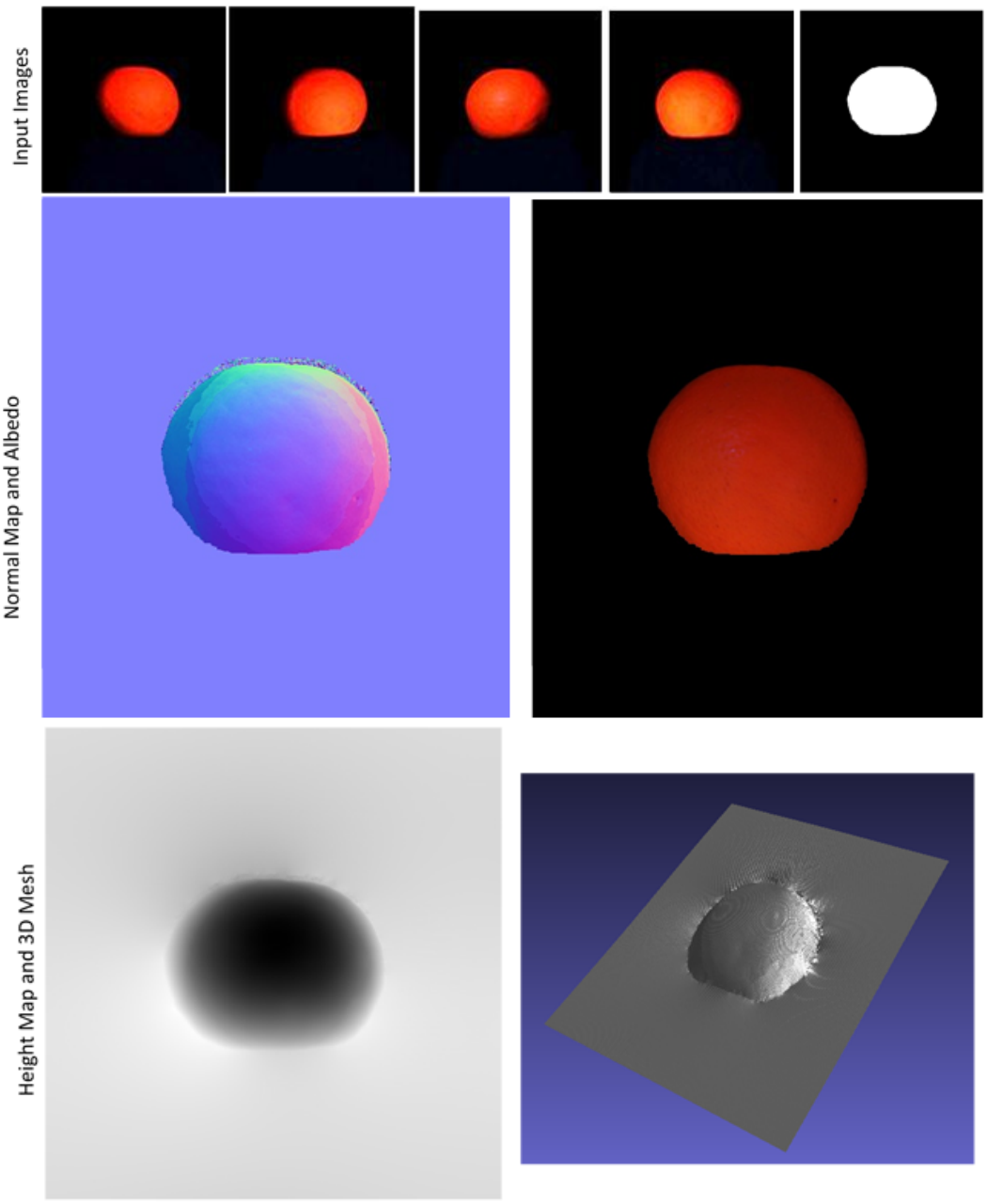


Figure 6.14: Results for Orange object under Polarised IColor MR Lamp setting

Here we observe a detailed surface normal along with some noise at the top of an orange. The 3D mesh is smoother and details out some bumps. The 3D Mesh also has noise at the top of the orange which is expected as per some noise during normal map calculation. The albedo on the other hand clearly shows the total reflectivity property for the orange.

Infrastructure type
: - Canon Camera with IColor MR Lamp
: - Red Banks i.e. Max Value for Red, 0 for Green and 0 for Blue channels

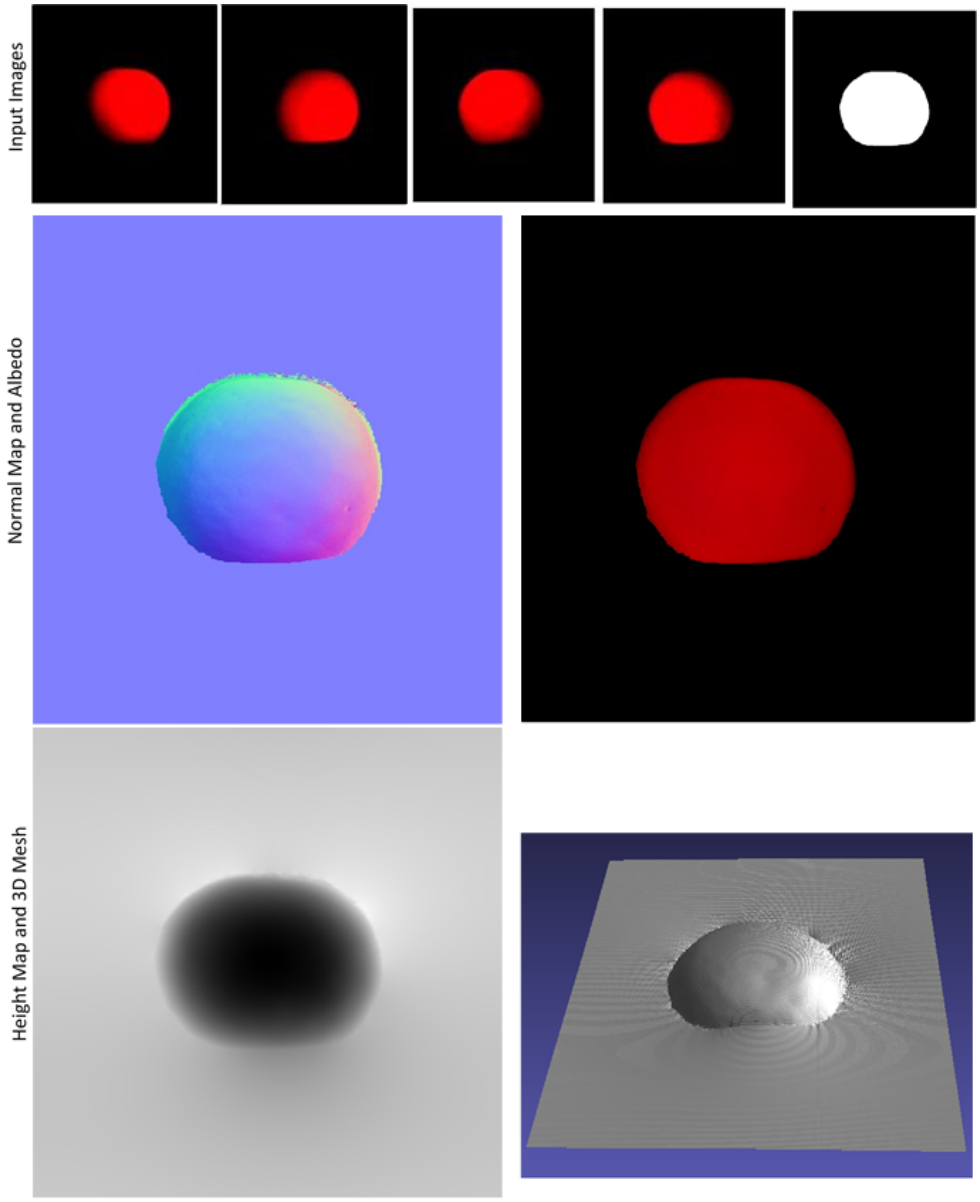


Figure 6.15: Results for Orange object under Polarised IColor MR Lamp setting

The normal map observed has less surface details. However it has very less of the noise and the 3D mesh obtained has very little details on small bumps orange possess. It can be understood that the Red light scattered mostly in the orange and thus the total reflectivity is minimal because of less reflectance data captured.

Infrastructure type
: - Canon Camera with IColor MR Lamp
: - Green Banks i.e. Max Value for Green, 0 for Red and 0 for Blue channels

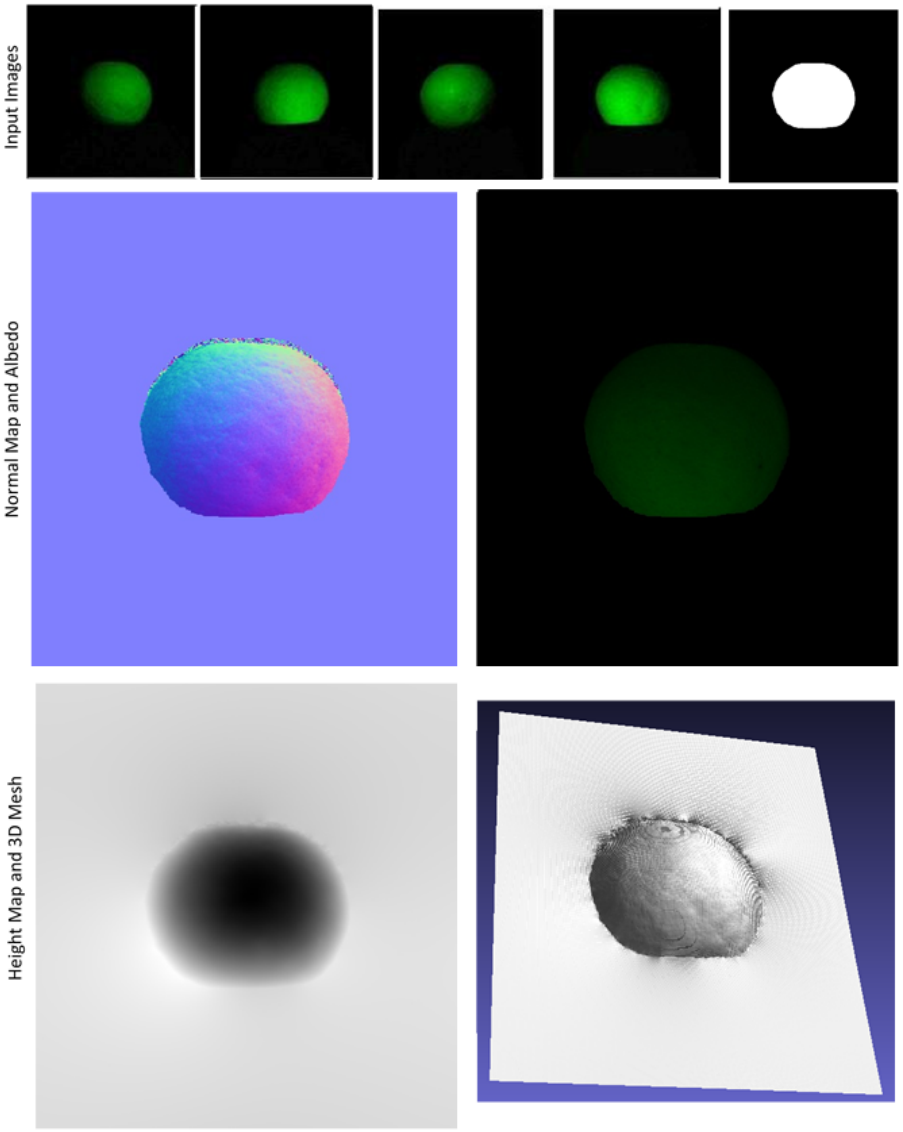


Figure 6.16: Results for Orange object under Polarised IColor MR Lamp setting

From the above three figures it can be seen that the surface normal computed with white LED's (RGB full on) has more surface details than the surface normal computed with red light. It is understood that the red light scatters the most in the orange and therefore gives less of the surface details. Similarly, in comparison to the red light, green light results in slightly higher surface details but rather less surface details to that of white (RGB full on) lamps.

6.2.3 With IColor MR white lights

Let us now consider another experiment which involves a different setup. Firstly IW MR Lamps with its warm, neutral and cold channels to maximum intensities. Secondly with IColor lamps with its warm (only) channel to maximum intensity and finally with its cool channel set to maximum intensity. The object is an orange for this experiment

The light direction for the following three setups is shown below.

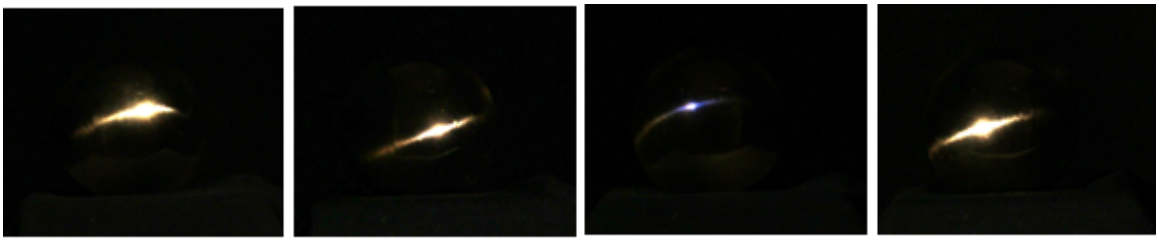


Figure 6.17: Light Direction for IW MR white lights as reflected in the mirror ball

Infrastructure type
: - Canon Camera with IW MR White lamp
: - Full beam i.e. Max value for Warm, Neutral and Cool channels

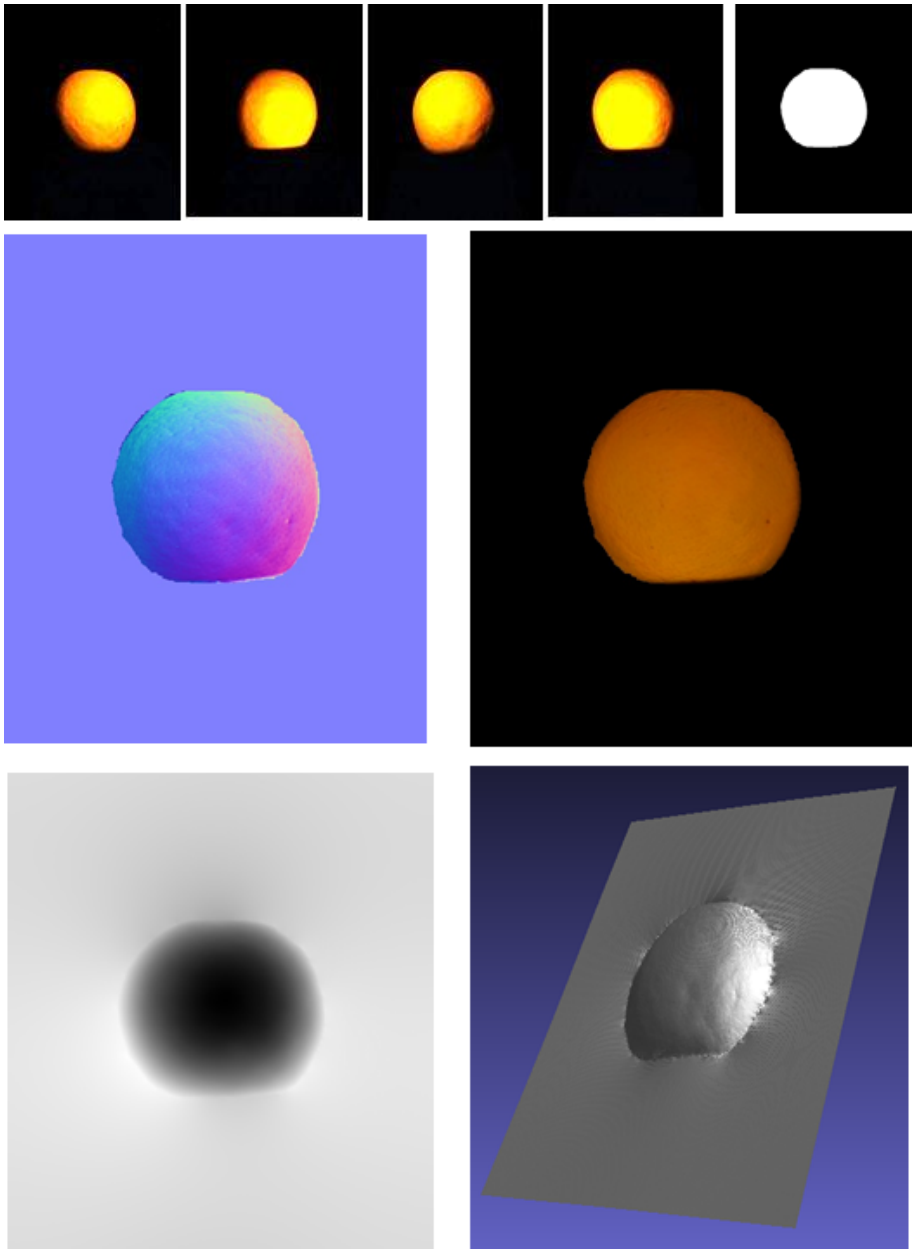


Figure 6.18: Results for Orange object under Polarised IW MR White Lamp setting

Here we can see the full beam warm, neutral and cool temperature lights produces rather distinct result. The noise is very minimal to any of the results obtained so far. The normal map has comparatively higher surface details and thus a rather polished 3D mesh is recovered.

Infrastructure type
: - Canon Camera with IW MR White lamp
: - Warm Banks i.e. Max Value for Warm, 0 for Neutral and 0 for Cool channels.

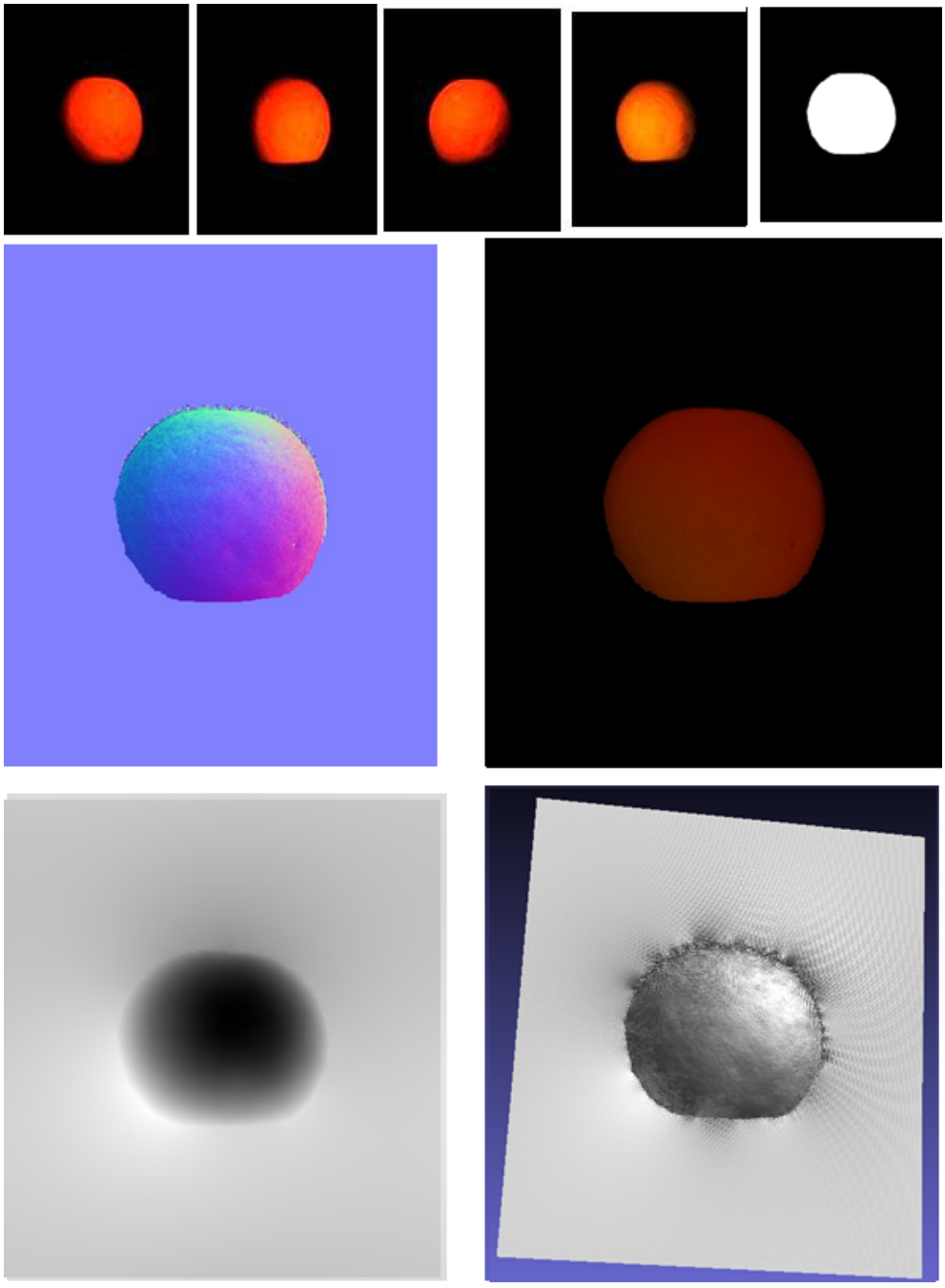


Figure 6.19: Results for Orange object under Polarised IW MR White Lamp setting

Here with warm banks, although higher details on orange is recovered as can be seen with the bump details observed, it seems the result is rather noisy on the peripheral. This reflects in the 3D mesh where least smooth outlining is observed.

Infrastructure type
: - Canon Camera with IW MR White lamp
: - Cold Banks i.e. Max Value for Cool, 0 for Neutral and 0 for Warm channels.

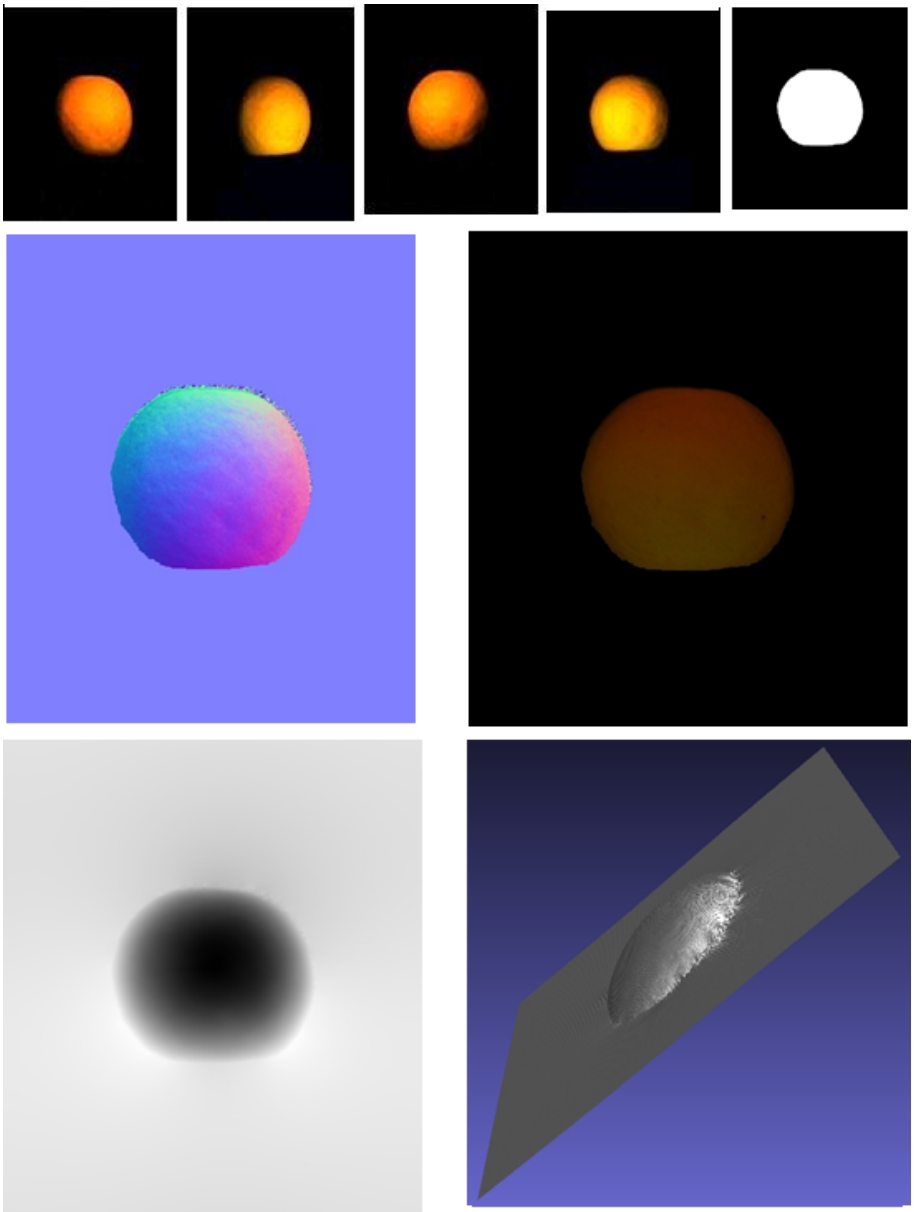


Figure 6.20: Results for Orange object under Polarised IW MR White Lamp setting

From the above three figures it can be seen that the surface normal computed with warm LED produces less details in the surface normal for the orange object compared to all the white and cold white LED's.

6.2.4 Experiment with narrow band RGB illumination

So far we performed photometric experiments with four different images obtained from four different lighting conditions. We realised how static the object under the experiment had to be for the right results i.e. each of four images had to be obtained without the object moving. This gave us the limitation to non-static objects. Thus being able to perform the entire experiment with just one image rather than four separate images was analyzed.

The next set of experiment performed was with an egg object. The experiment made was done by using just three of the four RGB lights where we illuminate the object with one red light, one green light and one blue light from three different directions. Then a single photo of the egg lit by these three lights was used to compute a surface normal map by using the red channel of the camera for the red light, green channel of the camera for the green light and blue channel of the camera for the blue light. However, the albedo map could not be calculated using just one picture as described but was later deduced with a second photograph with all RGB lights on to the object.

The three lighting direction used for the experiment is shown below.

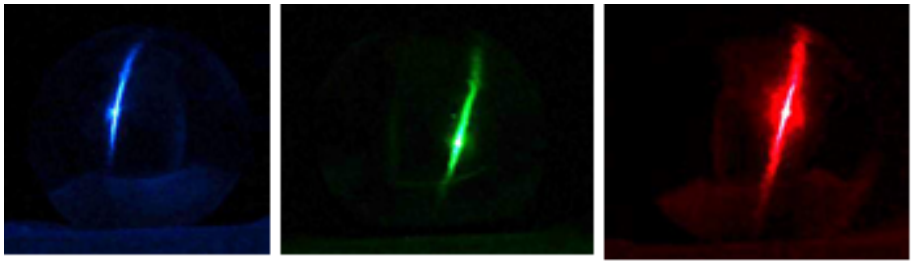


Figure 6.21: Three different (RGB) lighting direction

With varying lights i.e. Red, Green and Blue from three different direction setup, we obtain a single image when all three lights are at its full intensities. The following image on the left is obtained. Similarly the mask (right) is recovered for the same image for further photo-metric experiment.

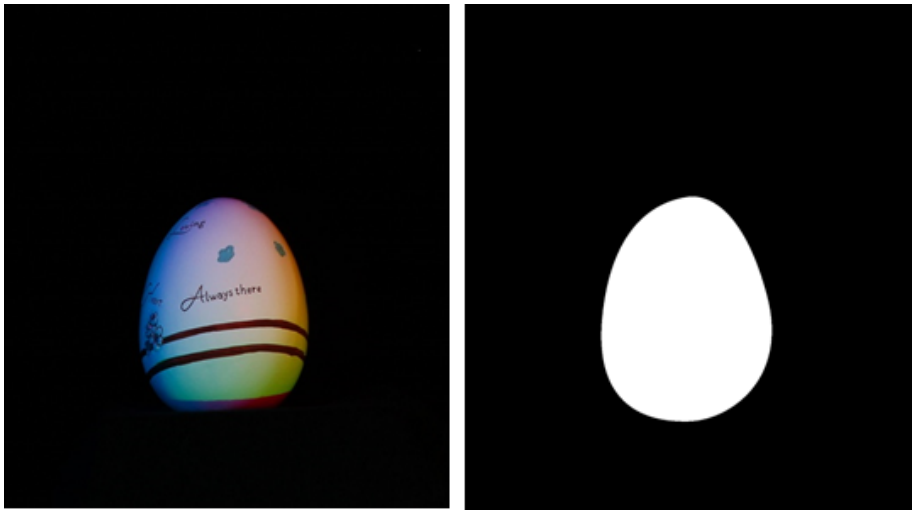


Figure 6.22: Single image(left) obtained from the setup along with the recovered mask(right).

As the single obtained image would have three separate channels with Red, Green and Blue intensities along with three separate lighting directions. We can thus use the same C++ program discussed earlier to recover the normal map and 3D mesh shown below.

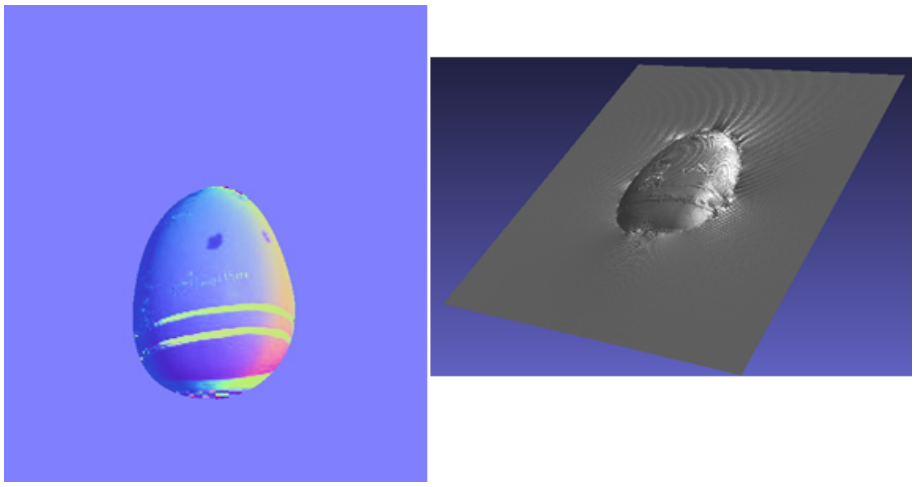


Figure 6.23: Normal map(left) with 3D Mesh (right).

The observed normal map above seem to have artifacts which is understood to have caused due to noise in channels. Also in earlier experiments we had four separate images i.e. more data set to analyze and reduce noises. However with a single image in this experiment, we could not reduce noises. Similarly the 3D mesh seen is not smoothly plotted due to some erroneous data in normal map. Also it can be observed that the overall surface detail has been recovered well and we can therefore make a conclusion that the experiment with narrow band RGB illumination is valid.

7 | Evaluation

We have so far evaluated the performance of two of our setups by thoroughly comparing and analyzing the results. We deduced the only way to check if the setups built had a desired outcome was to do so visually with output images observed. In terms of implementation perspective we compared the output between expected results obtained from verified source and our experiment result, which was done as presented with normal map comparison of object Buddha in Chapter 6 (Figure 6.7).

Similarly, in previous Chapter, while discussing the various results obtained under varying setups, we performed visual analysis on the kind of output obtained. We talked about the differences in results along with the possible cause and also the technicalities leading to some erroneous data. We made some evaluation on choice of setup in Chapter 3 when deciding on the type of flashes to use with regard to its synchronisation possibilities and ease of use. Further evaluation was carried out when choosing the right resources for the project setup in Chapter 5 i.e. when choosing the right LED lighting system, where we analysed various beam angles lamps in terms of its polar candela distribution, lumens with efficacy and illuminance properties.

However to give more context to the obtained results, we decided to further evaluate the generated system and its usefulness in real world scenario, and thus decided to use one the verified software (named Renderer) written entirely by a PHD student at Imperial College London to validate the results obtained so far with our experimental setup.

The software named Renderer is a Real-time Rendering system implemented in Python with use of OpenGl and GLSL libraries. It was written by a PHD student Jeremy Riviere who is currently working as a team member for Realistic Graphics and Imaging group [34] responsible for conducting research in realistic computer graphics spanning acquisition, modelling and rendering real world materials. The following is the visual user interface for the software.

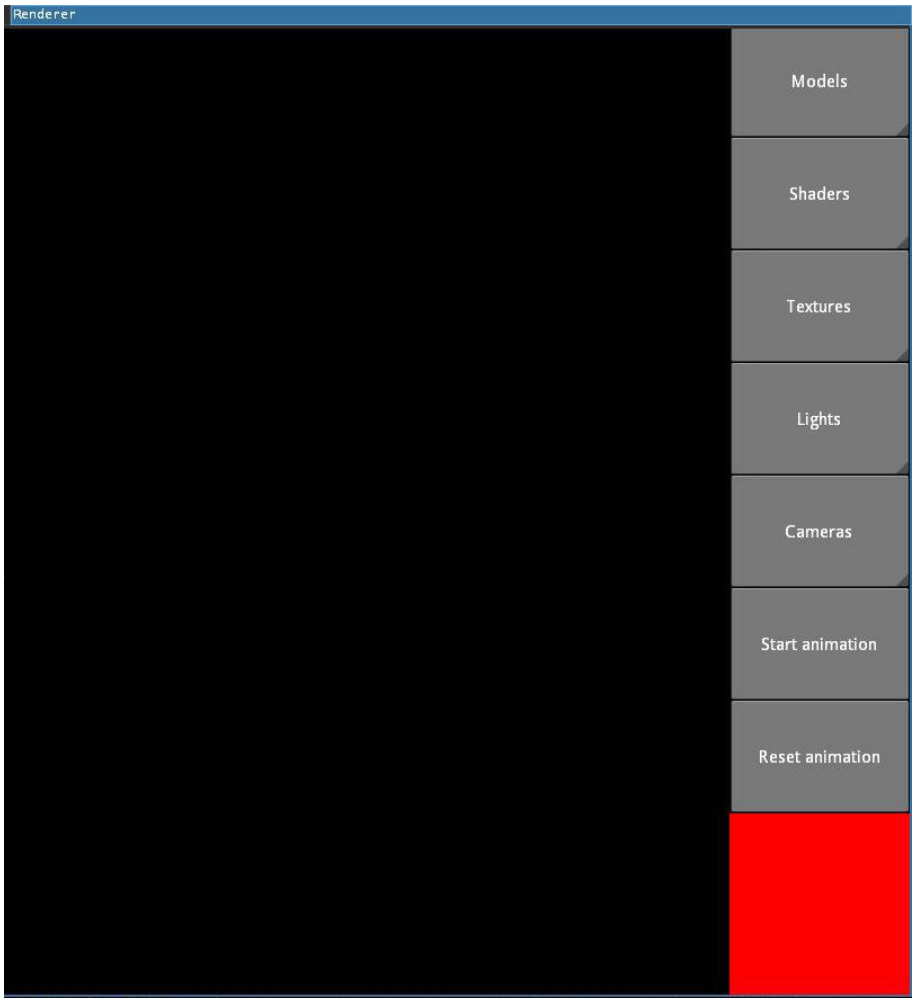


Figure 7.1: Realtime Rendering system by Jeremy Riviere

With many more functionality offered by the Renderer software, we will be using its specific capability of taking a normal map and Albedo as input to render an image under infinite lighting directions. We will then compare the results obtained from our setup at a certain light direction against the result obtained from the Renderer software to the same lighting direction.

The following shows the use of the software for an orange object with input of normal map and albedo producing a rendered image under following light direction.

X = -0.498408
Y = 0.330270
Z = 0.801565

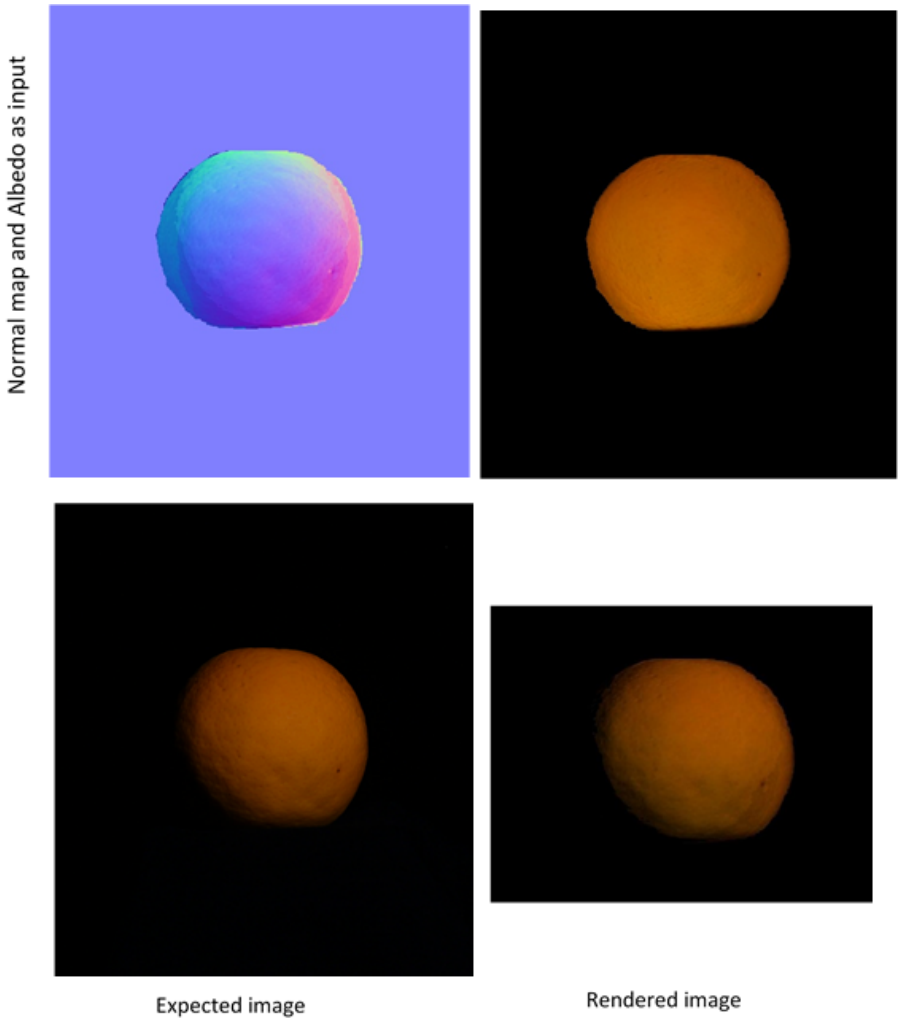


Figure 7.2: Obtained result for orange object using Renderer software

Similarly, the following shows the use of the software for an egg bject with input of normal map and albedo producing a rendered image under following light direction.

X = -0.432004
Y = -0.333593
Z = 0.837907

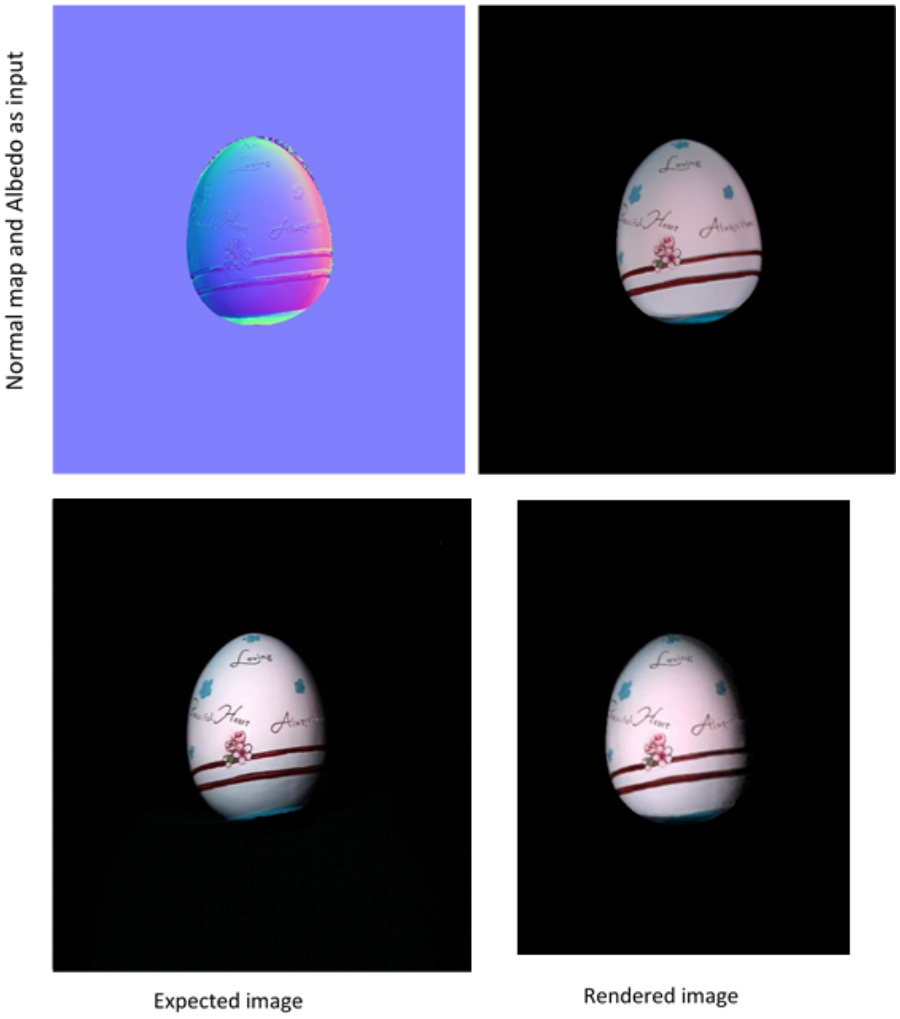


Figure 7.3: Obtained result for egg object using Renderer software

As from both set of example given in Figure (7.2) and Figure (7.3) we see that the expected image is identical to that of the rendered image. For the kind of inputs we got precisely the result we would have hoped for as the shadow effects and the shape has been rendered as expected for both egg and orange object. We are particularly pleased with the results for both egg and orange as there is qualitatively a good match visually. Since we used a pre-approved software to make the comparison on results and the output was a good match, we evaluate that the implementation was successfully done and the setup was rightly built.

8 | Conclusion

We set out to build programmable reflectance acquisition setups using DSLR cameras and controllable light sources for realistic graphics applications. We started off with general research on Photometric Stereo techniques and its use in obtaining surface normal. We derived a formula and also mentioned the normal map integration to understand the 3D reconstruction process. Finally, with the background research on photometric stereo, we required to build a system to put photometric experiment into action. Thus we went onto building two set of systems which involved Canon camera with Speedlite flash lights and Colour Kinetic lighting systems. We performed various experiments under these setups to obtain results for comparison. Finally, we wanted to verify the working system and thus put out various implementations and testing procedures.

Firstly we implemented the API provided by canon to program DLSR camera to retrieve functionality that controlled the camera. The controlled camera allowed many functions applicable for the experiments for example the images taken could be retrieved to the host computer by programming the DSLR camera. Also with an introduction of wireless flashes and a mechanism to control them, it soon became possible to perform photometric stereo experiment under this setup. With a simple setup we were able to obtain four images with fixed camera position but with varying light sources similar to the photometric stereo technique.

Similarly, various research was done in understanding the existing light stages. Variants of light stages were discussed in order for us to rebuild similar setups and scale it up in near future while understanding the underlying principle in obtaining relevant data. Thus we set out to build an initial version of light stage and did so successfully with the use of programmable colour kinetic system. We built a system by setting up a programmable network of LED lights from Philips Colour Kinetics. For the complete system to link together we went onto building our own Python/C++ based API through reverse engineering data transfer in a network. The API was then used in controlling the LED lights while also extending it to control RGB and white LED lights over an Ethernet network. In addition it soon became apparent to test the system which was gained by performing photometric experiments using these setup which later extended to an experiment with narrow band RGB illumination.

The two setups built throughout the project allowed us to perform similar experiments and thus a comparison was done on the practicality of the two system for future usage. It was soon realised both setup would be useful in many other application in near future. The project still at its extension phase, we believe the system can be further extended as per its initial implementation. Thus the possible future work would be to build an entire light stage based on the framework we have developed with LED lights.

On the whole, we are happy with what we have achieved throughout the project as it also unfolds the possibilities of extending the project. As a result of this, the setup built was tested and verified with various experiments performed as we set out to produce the whole system that was usable.

APPENDIX A

A.1

```
//-----------------------------------------------
//converts given string to hex value
string UTILS::string_to_hex(char* input)
{
    static const char* const lut = "0123456789ABCDEF";
    size_t len = strlen(input);
    if (len & 1) throw std::invalid_argument("odd length");

    std::string output;
    output.reserve(len / 2);
    for (size_t i = 0; i < len - 1; i += 2)
    {
        char a = input[i];
        const char* p = std::lower_bound(lut, lut + 16, a);
        if (*p != a) throw std::invalid_argument("not a hex digit");

        char b = input[i + 1];
        const char* q = std::lower_bound(lut, lut + 16, b);
        if (*q != b) throw std::invalid_argument("not a hex digit");

        output.push_back(((p - lut) << 4) | (q - lut));
    }
    return output;
}
```

Figure 8.1: C++ code for string to Hexadecimal Conversion

```
EdsError Canon650DCamera::setShutterSpeed(EdsCameraRef camera , EdsUInt32 valueShutterSpeed)
{
    EdsPropertyDesc* TvValue = new EdsPropertyDesc();
    EdsError err = EDS_ERR_OK;
    EdsDataType dataType ;
    EdsUInt32 dataSize;

    EdsChar * t = new EdsChar[];
    err = EdsGetPropertySize(camera, kEdsPropID_Tv, 0, &dataType, &dataSize);
    EdsUInt32* shutterSpeedDetail = new EdsUInt32[dataSize];

    if (err == EDS_ERR_OK)
    {
        err = EdsGetPropertyData(camera, kEdsPropID_Tv, 0, dataSize, shutterSpeedDetail);
        if (err == EDS_ERR_OK){
            detailOutCurrentShutterSpeed(shutterSpeedDetail);
        }
    }

    err = EdsSetPropertyData(camera, kEdsPropID_Tv, 0, sizeof(valueShutterSpeed), &valueShutterSpeed);
    if (err == EDS_ERR_OK)
    {
        cout << "Shutter Speed Correctly set";
    }
    return err;
}
```

Figure 8.2: Shutter speed program using C++ and canon EDSDK API

A.2

VTK toolkit

Figure 8.3: VTK Logo[35]

The Visualization toolkit (VTK) is an open source software system useful for image processing and 3D computer graphics. It consists of several interpreted interface layers for example Java, Tcl/Tk, Python and also consist of C++ libraries. With its C++ implementation to provide wide variety of visualization algorithms, 3D rendering was a possibility using VTK toolkit. This therefore has become a useful tool in our project as it allowed us to visualize 3D Mesh obtained from the photometric stereo experiment.

The following shows the display function which makes extensive use of VTK tool kit functionality to show 3D mesh.

```
void Display3DMesh::display() {
    /* The pipeline the vtk toolkit follows is vtkPoints -> vtkPolyData -> vtkPolyDataMapper
                                                                    -> vtkActor -> vtkRenderer */
    vtkSmartPointer<vtkPoints> points = vtkSmartPointer<vtkPoints>::New();
    vtkSmartPointer<vtkPolyData> polyData = vtkSmartPointer<vtkPolyData>::New();
    vtkSmartPointer<vtkPolyDataMapper> modelMapper = vtkSmartPointer<vtkPolyDataMapper>::New();
    vtkSmartPointer<vtkActor> modelActor = vtkSmartPointer<vtkActor>::New();
    vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
    vtkSmartPointer<vtkCellArray> vtkTriangles = vtkSmartPointer<vtkCellArray>::New();

    // inserting the x,y z to our initialized points object /
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            points->InsertNextPoint(x, y, integratedGlobalHeightZ.at<float>(y, x));
        }
    }

    /* Now we bring connection to grid points */
    vtkSmartPointer<vtkTriangle> triangle = vtkSmartPointer<vtkTriangle>::New();
    triangle->GetPointIds()->SetNumberOfIds(3);
    for (int i = 0; i < height - 1; i++) {
        for (int j = 0; j < width - 1; j++) {
            triangle->GetPointIds()->SetId(0, j + (i*width));
            triangle->GetPointIds()->SetId(1, (i + 1)*width + j);
            triangle->GetPointIds()->SetId(2, j + (i*width) + 1);
            vtkTriangles->InsertNextCell(triangle);
        }
    }
    polyData->SetPoints(points);
    polyData->SetPolys(vtkTriangles);
    modelMapper->SetInputData(polyData);

    /* render mesh */
    vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::New();
    vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer<vtkRenderWindowInteractor>::New();
    interactor->SetRenderWindow(renderWindow);
    renderWindow->AddRenderer(renderer);
    renderWindow->Render();
    interactor->Start();
}
```

Figure 8.4: C++ with VTK to display 3D Mesh

OpenCV

OpenCv namely Open source Computer Vision is an open source cross platform library that allows real time image processing. With its computational efficiency and multi-core processing capabilities, it takes advantage of hardware acceleration during image processing. As its primary interface is in C++, it was applicable for use in our application.

Figure 8.5: Light Stage 5[36]

The following code written in C++ to obtain Height map from x and Y gradients obtained while deducing normal map, makes extensive use of OpenCV functions.

```
cv::Mat PhotometricStereo :: obtainHeightMap(cv::Mat Pgrads, cv::Mat Qgrads) {

    cv::Mat P(Pgrads.rows, Pgrads.cols, CV_32FC2, cv::Scalar::all(0));
    cv::Mat Q(Pgrads.rows, Pgrads.cols, CV_32FC2, cv::Scalar::all(0));
    cv::Mat Z(Pgrads.rows, Pgrads.cols, CV_32FC2, cv::Scalar::all(0));

    float lambda = 1.0f;
    float mu = 1.0f;

    //Discrete Fourier Transform that decomposes an image into its sine and cosine components.
    cv::dft(Pgrads, P, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(Qgrads, Q, cv::DFT_COMPLEX_OUTPUT);
    for (int i = 0; i<Pgrads.rows; i++) {
        for (int j = 0; j<Pgrads.cols; j++) {
            if (i != 0 || j != 0) {
                float gradRow = sin((float)(i * 2 * CV_PI / Pgrads.rows));
                float gradCol = sin((float)(j * 2 * CV_PI / Pgrads.cols));

                float uv = pow(gradRow, 2) + pow(gradCol, 2);
                float d = (1.0f + lambda)*uv + mu*pow(uv, 2);
                Z.at<cv::Vec2f>(i, j)[0] = (gradRow*P.at<cv::Vec2f>(i, j)[1] + gradCol*Q.at<cv::Vec2f>(i, j)[1]) / d;
                Z.at<cv::Vec2f>(i, j)[1] = (-gradCol*P.at<cv::Vec2f>(i, j)[0] - gradRow*Q.at<cv::Vec2f>(i, j)[0]) / d;
            }
        }
    }
    /* setting unknown average height to zero */
    Z.at<cv::Vec2f>(0, 0)[0] = 0.0f;
    Z.at<cv::Vec2f>(0, 0)[1] = 0.0f;

    cv::dft(Z, Z, cv::DFT_INVERSE | cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);
    cv::Mat heightMap(Pgrads.rows, Pgrads.cols, CV_32FC2, cv::Scalar::all(0));
    cv::normalize(Z, heightMap, 0, 255, cv::NORM_MINMAX);
    return Z;
}
```

Figure 8.6: C++ with opencv to obtain height map

APPENDIX B

B.1

```
IESNA:LM-63-2002
[TEST]ITL76007
[TESTLAB]INDEPENDENT TESTING LABORATORIES, INC.
[ISSUEDATE]02/07/13
[_REVISED]02/12/13
[MANUFAC]PHILIPS COLOR KINETICS
[LUMCAT]iCOLOR MR GEN3, 30-DEGREES, ALL ON
[LAMP]ONE MR-16 STYLE BI-PIN BASE LAMP, 2-PIECE CAST SILVER PAINTED
[MORE]METAL BODY, 7 LIGHT EMITTING DIODES (LEDS) - 3 RED, 2 GREEN, 2
[MORE]BLUE. CLEAR FLAT PRISMATIC PLASTIC LENS WITH ONE OPTIC BELOW
[MORE]EACH LED, FORMED SILVER PAINTED METAL LENS RETAINING RING/TRIM.
[MORE]LAMP NOT DISASSEMBLED FOR INSPECTION. LAMP AND LEDS VERTICAL
[MORE]BASE-UP POSITION.
[OTHER]TOTAL INPUT WATTS = 13.1 AT 120.0 VOLTS
[_LEDDRIVER]PHILIPS COLOR KINETICS PDS-70mr 24 (P/N: 109-000018-01),
[MORE]PHILIPS COLOR KINETICS iPLAYER3 (P/N: 103-000019-00)
[_NOTE]DATA SHOWN IS ABSOLUTE FOR THE SAMPLE PROVIDED AT RATED INPUT
[MORE]VOLTAGE (120VAC, 60Hz) TO THE LED DRIVER. A 10 FOOT POWER CORD
[MORE]EXTENDS FROM THE LED DRIVER TO THE LED ASSEMBLY.
[OTHER]TEST PROCEDURE: IESNA LM-79-08
[OTHER]TEST DISTANCE = 25.25 FEET
[_ABSOLUTELUMENS]151
TILT=NONE
7 -1 1 69 1 1 1 -0.125000 -0.125000 0.000000
1 1 13.1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10 10.5 11 11.5 12
 12.5 13 13.5 14 14.5 15 15.5 16 16.5 17 17.5 18 18.5 19 19.5 20 22.5 25 27.5
 30 32.5 35 37.5 40 42.5 45 47.5 50 52.5 55 57.5 60 62.5 65 67.5 70 72.5 75
 77.5 80 82.5 85 87.5 90
0
600 600 599 596 592 586 580 572 563 553 542 530 517 503 488 473 456 440 424 407
 390 374 357 340 323 307 291 275 259 244 229 215 202 189 175 164 152 141 130
 121 112 75 51 35 25 17 13 10 7 6 5 4 3 2 2 2 1 1 1 1 1 1 0 0 0 0 0 0 0
```

Figure 8.7: IColor MR gen3 30 degree beam angle details

```
IESNA:LM-63-2002
[TEST] 2,935
[TESTLAB] TUV SUD America
[ISSUEDATE] 7/22/2013 4:29:40 PM
[MANUFAC] Philips Color Kinetics
[LUMCAT] iW MR gen3, 27-DEGREES
[LUMINAIRE] IntelliWhite MR lamp with 6 LEDs and integrated optics
[LAMPCAT] PCK SKU 501-000015-01
[LAMP] JI1306642   817-1  SAMPLE 2A
[_AMBIENT] 24.6 deg C
[_POWERFACTOR] 0.8573
[_ATHD] 33.18%
[_CURRENT] 0.120
[_VOLTAGE] 119.96
[_STABILIZATION] 36:53.4 Stable
[_POWER VARIATION] 0.00 %
[_LIGHT VARIATION] 0.34 %
[_Lumens] 240.54
[Note 1] PCK SKU: 501-000015-01
[Note 2] all channels
TILT=NONE
1 -1 1.0 91 73 1 1 -0.124 -0.124 0
1.0 1 12.40
0.000 1.000 2.000 3.000 4.000 5.000 6.000 7.000 8.000 9.000 10.000 11.000 12.000 13.000 14.000
15.000 16.000 17.000 18.000 19.000 20.000 21.000 22.000 23.000 24.000 25.000 26.000 27.000 28.000 29.000
30.000 31.000 32.000 33.000 34.000 35.000 36.000 37.000 38.000 39.000 40.000 41.000 42.000 43.000 44.000
45.000 46.000 47.000 48.000 49.000 50.000 51.000 52.000 53.000 54.000 55.000 56.000 57.000 58.000 59.000
60.000 61.000 62.000 63.000 64.000 65.000 66.000 67.000 68.000 69.000 70.000 71.000 72.000 73.000 74.000
75.000 76.000 77.000 78.000 79.000 80.000 81.000 82.000 83.000 84.000 85.000 86.000 87.000 88.000 89.000
90.000
```

Figure 8.8: IW MR 26 degree beam angle details

B.2

Additional experimental results

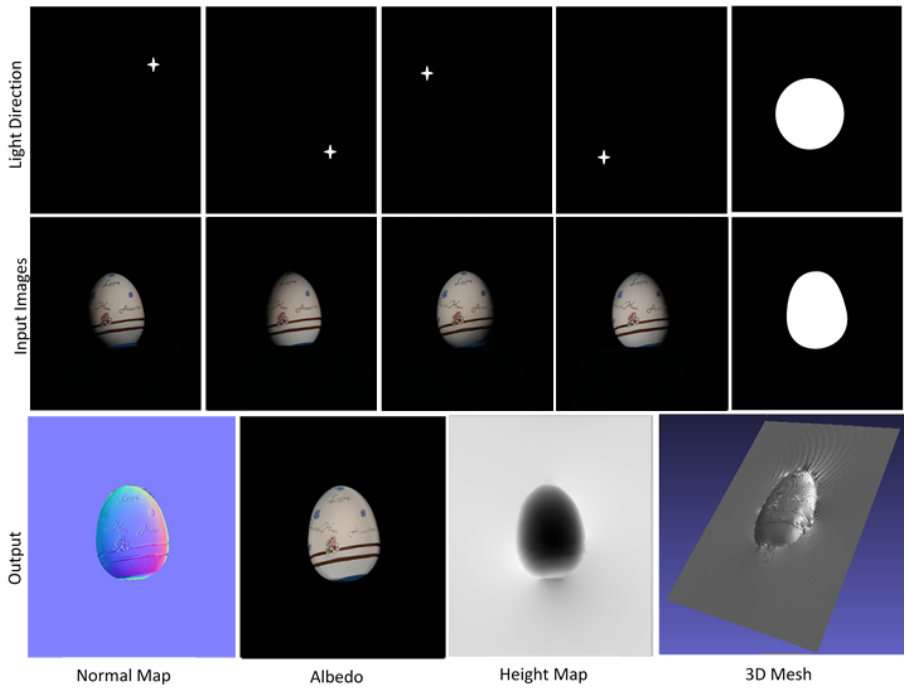


Figure 8.9: Egg under IW MR white with all banks set to full intensities

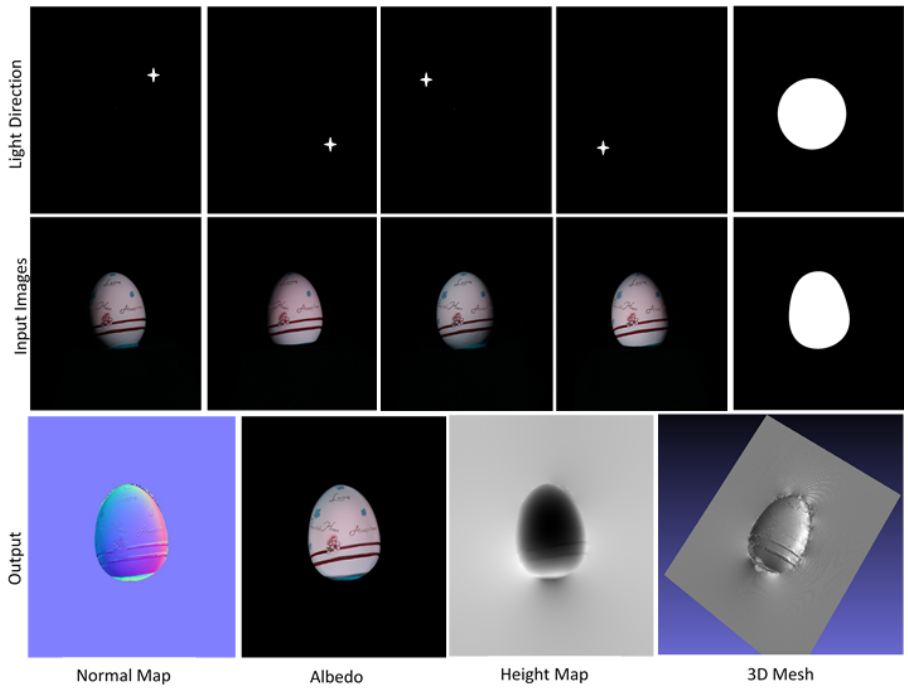


Figure 8.10: Egg under IColor RGB lights with all channels set to full intensities

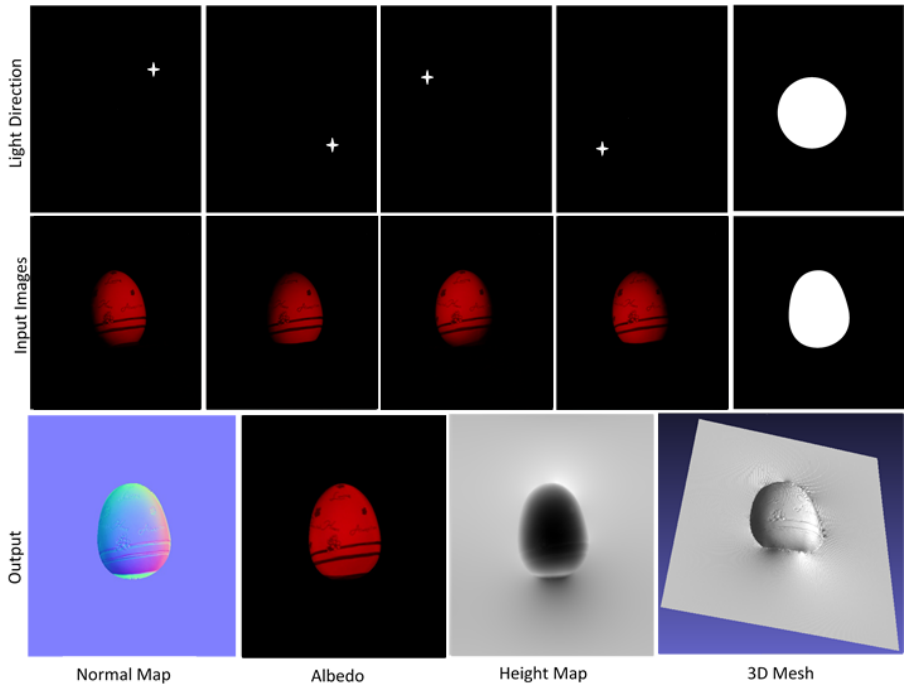


Figure 8.11: Egg under IColor lights with only Red channel set to full intensity

Bibliography

[1] Ronen Basri, David Jacobs, and Ira Kemelmacher. Photometric stereo with general, unknown lighting. *Int. J. Comput. Vision*, 72(3):239–257, May 2007.

[2] Paul Debevec. The light stages and their applications to photoreal digital actors. *SIGGRAPH Asia Technical Briefs*, 2012.

[3] Carlos Hernández, George Vogiatzis, Gabriel J Brostow, Björn Stenger, and Roberto Cipolla. Non-rigid photometric stereo with colored lights. pages 1–8, 2007.

[4] Berthold KP Horn and Robert W Sjoberg. Calculating the reflectance map. *Applied optics*, 18(11):1770–1779, 1979.

[5] Aditi Majumder, David Jones, Matthew McCrory, Michael E Papka, and Rick Stevens. Using a camera to capture and correct spatial photometric variation in multi-projector displays. In *IEEE International Workshop on Projector-Camera Systems*, 2003.

[6] Wojciech Matusik. A data-driven reflectance model. 2003.

[7] Mohammad Mirzadeh, Maxime Theillard, and Frédéric Gibou. A second-order discretization of the nonlinear poisson-boltzmann equation over irregular geometries using non-graded adaptive cartesian grids. *J. Comput. Phys.*, 230(5):2125–2140, March 2011.

[8] Shree K Nayar, Katsushi Ikeuchi, and Takeo Kanade. Shape from interreflections. *International Journal of Computer Vision*, 6(3):173–195, 1991.

[9] Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental analysis of brdf models. *Rendering Techniques*, 2005:16th, 2005.

[10] Vincent Nozick. Pyramidal normal map integration for real-time photometric stereo. In *EAM Mechatronics 2010*, pages 128–132, 2010.

[11] N. Petrovic, I. Cohen, B. J. Frey, R. Koetter, and T. S. Huang. Enforcing integrability for surface reconstruction algorithms using belief propagation in graphical models. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 1:743–748, 2001.

[12] Mike Seymour. The art of digital faces at ict - digital emily to digital ira. 2013, November 25.

[13] Robert J Woodham. Photometric method for determining surface orientation from multiple images. *Optical engineering*, 19(1):191139–191139, 1980.

[14] Chris Wynn. An introduction to brdf-based lighting. *Nvidia Corporation*, 2000.