Università
Ca'Foscari
Venezia

Department of
Environmental Sciences, Informatics and Statistics

Master's Degree Programme in
COMPUTER SCIENCE

An Implicit Neural Representation for Reflectance Transformation Imaging (RTI)

Supervisors:
Prof. Filippo Bergamasco
Dr. Mara Pistellato

Candidate:
Sandro Baccega
ID. 865024

Academic Year 2023-2024

Contents

Abstract

Reflectance Transformation Imaging (RTI) is a widely used method for obtaining detailed per-pixel reflectance data by photographing an object under varying lighting conditions. The gathered information can then be utilized to re-light the subject interactively and reveal surface details that would be impossible to see with a single photo. In this thesis, we propose a data-driven approach based on an Implicit Neural Representations (INR) of the light transport function to interactively relight the scene in a photorealistic way. Comparisons with existing state-of-the-art methods demonstrate the feasibility of the approach and suggest further investigation of INRs for RTI applications.

Chapter 1

Introduction

Reflectance Transformation Imaging (RTI) is a popular Computational photography technique which allows to detect the details of the object's surface starting from a set of images. The technique, in fact, can reveal information about the rich representations of surfaces including geometric details and reflective behavior of materials using an algorithmic rendering that uses multiple known images of the object [3]. In order to retrieve such information, the set of images should be captured from a fixed point of view with varying light direction. This technique was first conceived in 2001 at Hewlett Packard Labs and, was originally called Polynomial Texture Maps (PTM) [36].

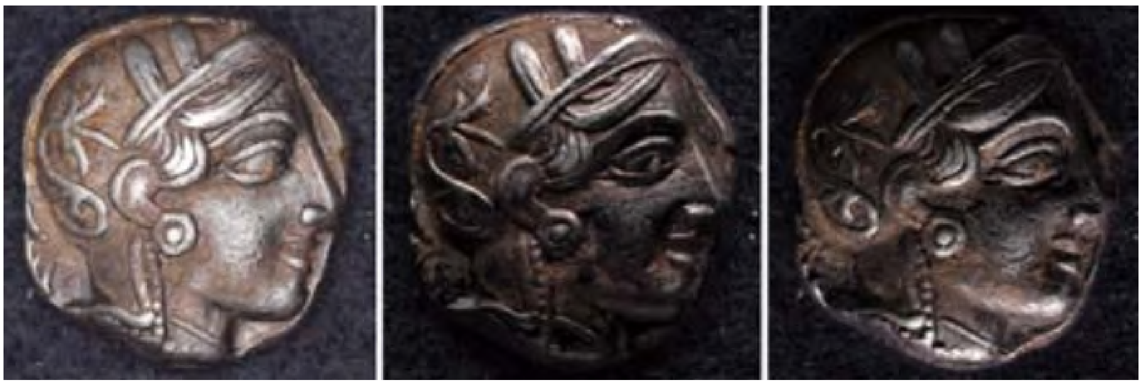
Reflectance Transformation Imaging is widely used in the cultural heritage sector due to its practical applications, such as material quality analysis of the surface, enhancement of visualization and consequently the production of relightable images. It is also useful for documentation and preservation, since it is possible to reconstruct the surface of damaged objects. In fact, the resulting images can reveal information that would otherwise be difficult to see with naked eyes, such as manufacturing techniques, surface conditions or conservation treatments. Regarding this benefit, it is necessary to underline the non-invasive characteristic of RTI. During the process of acquisition, in fact, materials and objects do not require to be touched and are not exposed to any harmful radiation.

Moreover, RTI is a flexible technique, so it can be applied to several materials and sizes, and it can be used in different environments, not only in a laboratory but also in open space. Because of its advantages, it can be complementary to or beneficial to other 3D models techniques such as photogrammetry or multispectral imaging. The main difference with Photogrammetry is that in the photogrammetry process, light and object are static while it is the camera that rotates; instead, the RTI process requires that the object and camera are static and the light source rotates. Moreover, RTI does not produce quantitative metric data, like photogrammetry, and, however it produces qualitative surface data, RTI is not able to show what lies beneath the surface. Beyond these differences, photogrammetry and RTI share some characteristics, because both are computational photographic image-based techniques and both involve passive image acquisition. However, RTI is not a 3D technology because it does not produce 3D models, like photogrammetry; instead, it generates a 2D detailed mathematical map of the surface texture, based on the normal vectors [3].

1.1 How RTI works

The way RTI works is rather simple: the object and the camera are at a fixed position, the only thing that changes between the photographs is the position of light. The light sources are fixed at a constant radius from the object and they surround it at incremental angles, forming a dome or hemisphere of light positions. The user may also add some real-time changes during the process, such as zoom in and out, increase or decrease sharpness, contrast and other lights or filters [11]. After the acquisition of all the images, the software processes all of them and produces one composite image, which is possible to manipulate in order to reveal surface details. The final image, in fact, initially looks like a flat, normal photograph, but then, if it is viewed with an RTI viewer, the user can freely move the virtual light source around the image, for example, using a mouse. In simple words, when the users drag their mouse through the image, which contains light information, they are dragging their mouse over a hemispheric volume. As a consequence, when the mouse cursor is in the center of the light space area, the illumination is analogous to the sun at 'high noon', and when the mouse cursor is dragged toward the edge of the light space area, the sun approaches the 'horizon' [5], [Figure: 1.1] shows the same RTI scan under three different light directions.

Figure 1.1: Silver Athenian Tetradrachma, 449 BCE [5]

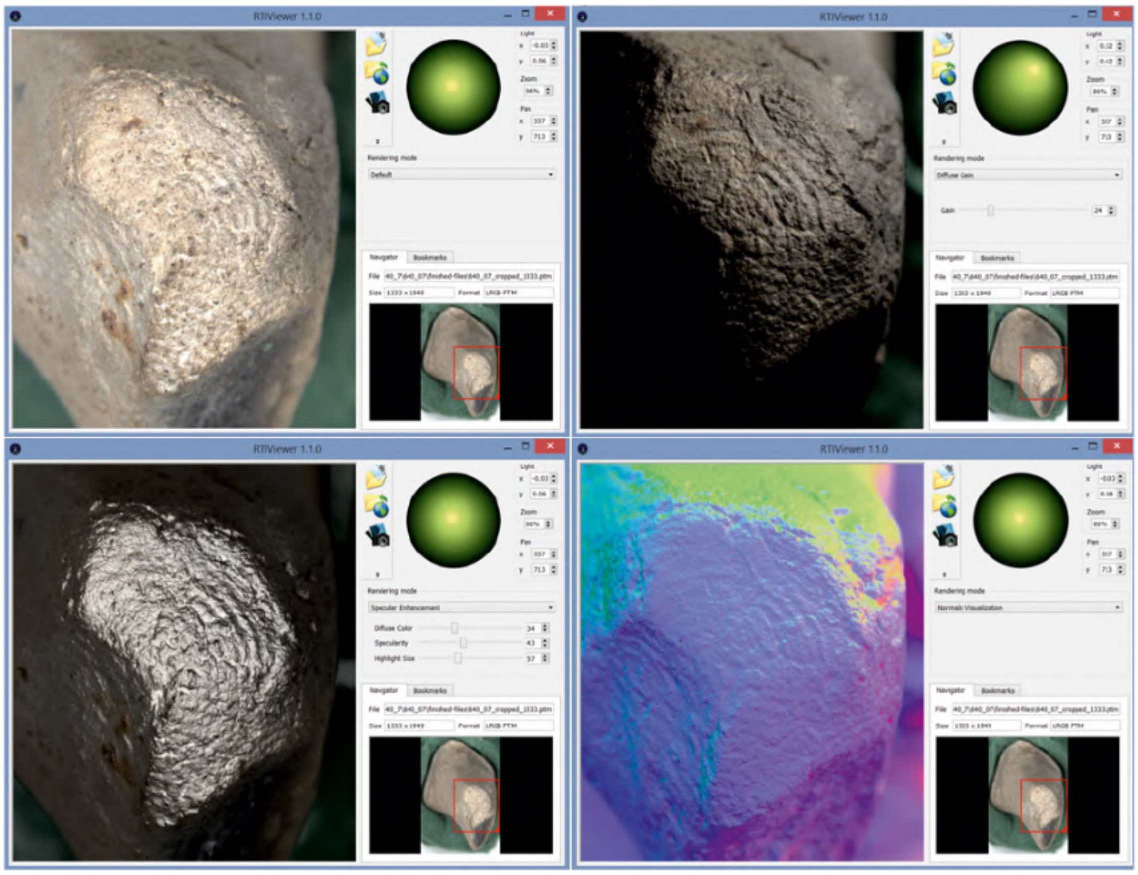


1.2 Applications of RTI

Reflectance Transformation Imaging it has been widely adopted in the cultural heritage sector, primarily because it requires only regular photographic hardware and some freely available software. In fact, the necessary equipment usually consists of only a hemispherical light dome with digital cameras on the top and lights in different positions to capture and enhance sample surface details, but, as the paper "On-the-go Reflectance Transformation Imaging with Ordinary Smartphones" [1] demonstrates, it is possible to use RTI with as little equipment as two smartphones and a printed marker, making it a very inexpensive method to capture this type of details.

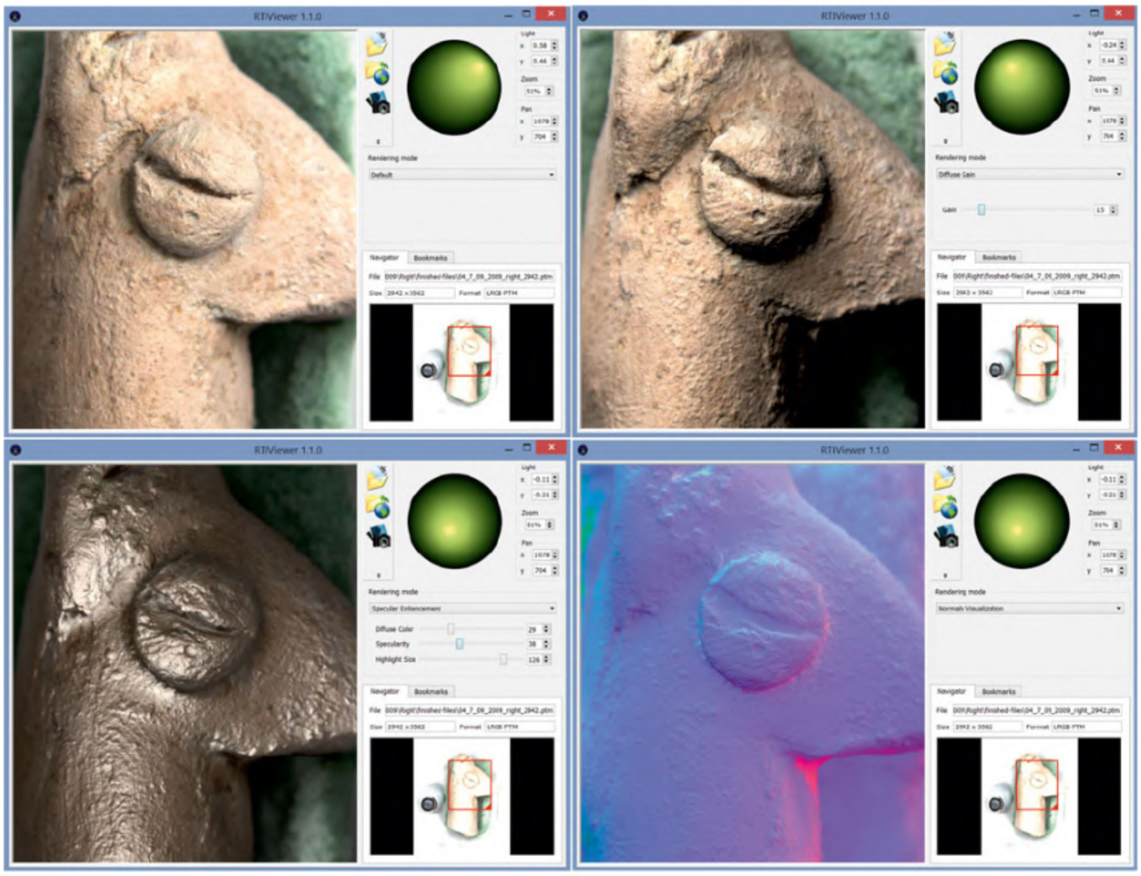
An example of application can be the Neolithic Figurines from Koutroulou Magoula in Greece [4]. In this project, RTI was capable of capturing the surface's marks, including fingerprints created unintentionally during the process of molding the figurines [Figure: 1.2], [Figure: 1.3].

Figure 1.2: Figurine 2012/640-07. Snapshot of the visualization through the RTI Viewer of the fingerprint identified onthe head of the figurine. Enhancement with computational algorithms in the following modes (from upper left to bottomright): Default, Diffuse Gain, Specular Enhancement, and Normals Visualization. [4]



Another example could be the analysis of the Folkton "Drums" [10] [Figure: 1.4], which are the most remarkable decorated artefacts from Neolithic Britain. The use

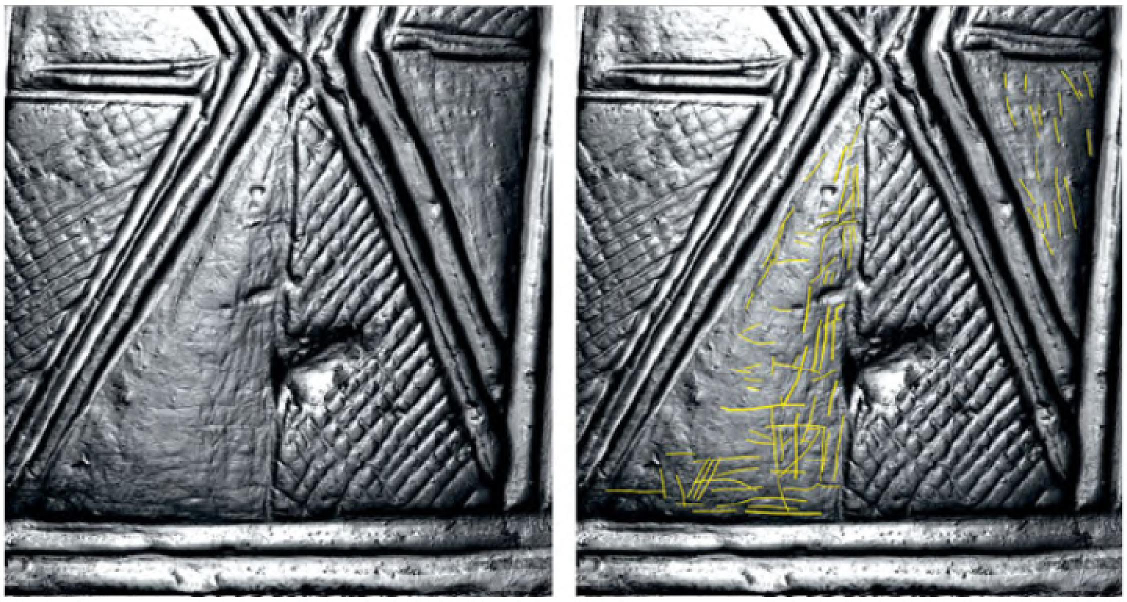
Figure 1.3: Figurine 2009/TH1-19. Snapshot of the visualization through the RTI Viewer of the partial fingerprintidentified on the right eye of the figurine and brush strokes on the hairdo, to the left of the eye. Enhancement withcomputational algorithms in the following modes (from upper left to bottom right): Default, Diffuse Gain, SpecularEnhancement, and Normals Visualization. [4]



of RTI has revealed evidence of the substantial erasure and consequent reworking of motifs on these objects. The decorations on these chalk drums, in fact, were carved and re-carved over time. There is no pre-ordained scheme, and this way of making artefacts and their decorations may have been very common in Neolithic Britain.

In another case RTI was used to study bone surface details, including cut marks, striations, etching and polishing [7]. The collection of manufactured bone artifacts, in this case study, derive from the ancient Maya site of El Zotz, Guatemala. Reflectance Transformation Imaging has proven to be effective for analyzing tech-

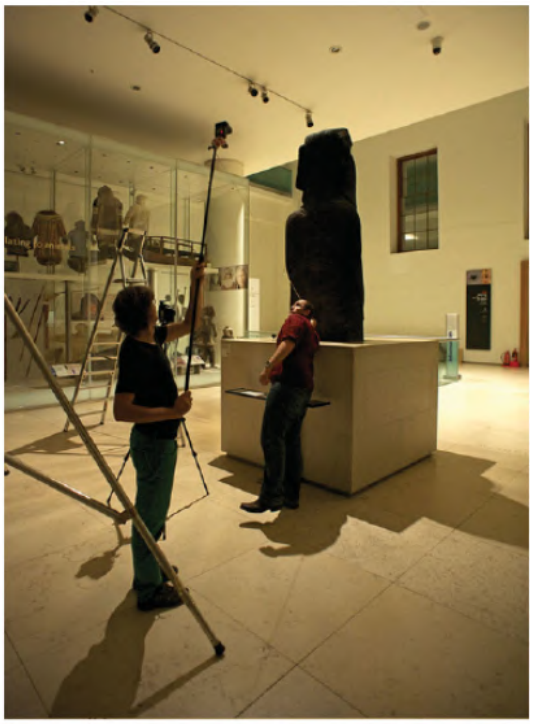
Figure 1.4: Reworking and erasure on the bottom blank space of the side panel, drum 1; erased motifs indicated in yellow;viewed under Reflectance Transformation Imaging Specular enhancement. [10]



nologies involved in the production of worked bones and investigating use-wear, also for examining post-depositional processes and conditions. Alterations found on the surfaces of archaeological bone specimens provide useful insights into past human behaviors and the natural processes that contribute to the formation of assemblages and archaeological sites. The physical marks resulting from manufacturing techniques and patterns of use-wear are useful in determining the technologies used to create bone tools and deducing their intended uses. Moreover, observable changes on bone surfaces can disclose taphonomic activities, including weathering, the effects of plant roots, damage from trampling, and the impact of gnawing by rodents or carnivores [7].

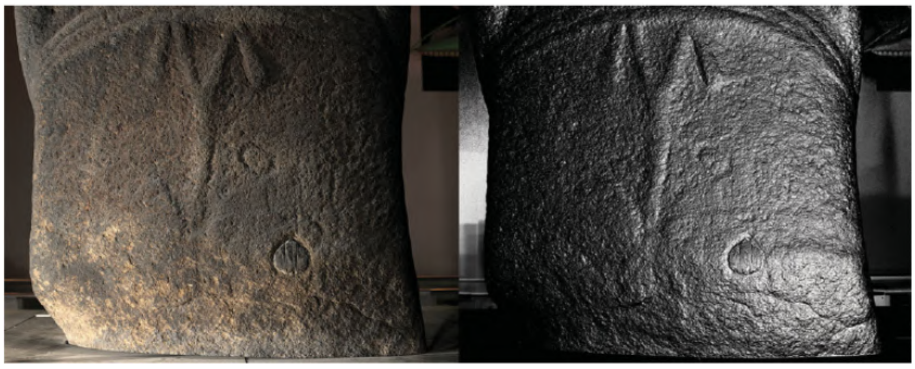
RTI has also been applied to objects of considerable dimension, like a statue that stands around 2.5m high on top of a 1.3m-high plinth. The statue in question was Hoa Hakananai'a, a fine Easter Island statue, which is displayed in the British Museum. Due to the dimensions of the statue, the team decided to split the back area into key sections, in order to have a better image resolution, otherwise the small details would not have been clear enough if the whole of the back was contained within a single image set. Moreover, there was the issue of the acquisition of the images. Since in RTI acquisition, the light source needs to be at the same distance from the object, scaffolding was required. It was necessary also to achieve a flat horizontal alignment, so to elevate the camera to the same height as the back of the head. This setup presented a challenge because the scaffolding offered limited positions for placing a camera tripod. Consequently, this adjustment introduced an additional RTI issue: camera focus. With the camera mounted on the scaffolding, focusing required a team member having to climb up and down between datasets. This activity occasionally resulted in minor camera movements, causing a sequence of out-of-focus shots [8]. [Figure: 1.5] shows an image of the capturing process.

Figure 1.5: RTI capture, with the camera fixed on scaffolding just out of shot to left; light is delivered by moveable remote flash, set at a constant distance from the statue determined by a length of string. [8]



After this process, it was possible to discover that the statue was made from the start with a tapering base, thanks to the study of the details of surface topography and condition at the lower extremity of the visible statue. In the past, it was suggested that the statue was meant to stand on a platform before it was moved to the site where it was found in 1868. [Figure: 1.6] shows an example of the results obtained by the research team.

Figure 1.6: Examples of an analysed RTI dataset of Hoa Hakananai'a (lower back), rendered in natural colours and with specular enhancement [8]



The need for high-resolution documentation of cultural heritage located beneath the water's surface is growing. For instance, Underwater RTI (URTI) has been used to capture details from historical shipwrecks in the Solent and the western Mediterranean [9]. This was the first adaptation of RTI protocols to the subaquatic environment, employing a scuba-deployable technique. The outcomes demonstrate that URTI is capable of revealing qualitative diagnostic details down to submillimeter precision on archaeological materials in their underwater settings [9]. [Figure: 1.7] shows the process in action.

1.3 Implicit Neural Representations

Implicit Neural Representations (INR) are types of neural networks that aim to approximate a continuous function by training on discrete examples. Since INRs provide continuous functions; they can, in theory, represent data at an infinitely fine resolution, making them a powerful tool in a wide range of applications, from image compression to audio processing.

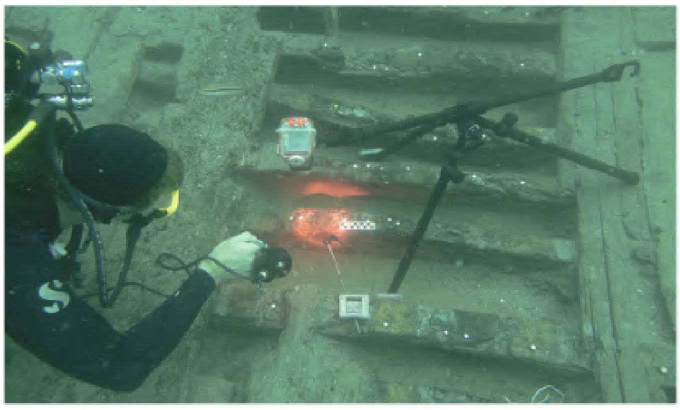
An example of INR are SIRENs [24], neural networks that use periodic activation functions capable of representing complex natural signals and their derivatives.

In our case, we are to approximate the interpolation function $f(x, y, \vec{l}) \rightarrow a$ by training a neural network Z with discrete examples (our training set), making our neural network with an INR.

To increase the accuracy of the model, since the input of our model are pixel coordinates and light directions, it is possible to use positional encoding [30]. This technique, that uses random Fourier features, maps the input coordinates to higher dimensional Fourier space $\gamma : R^D \rightarrow R^{2D}$ instead of giving them to the model directly. The mapping can be defined as [Eq: 1.1] where w is a randomly sampled gaussian matrix $N = (0, \sigma^2)$ of size D.

$$\gamma(v) \in \mathbb{R}^{2D} = \left[\cos(2\pi w_1^T v), \sin(2\pi w_1^T v), \ldots, \cos(2\pi w_D^T v), \sin(2\pi w_D^T v)\right]^T \tag{1.1}$$

Figure 1.7: Selmo performing URTI on the Cap del Vol, first century BC, Roman shipwreck. Photo credit: Dr. Gustau Vivar. [9]



Chapter 2

Related Work

This chapter will describe methods to capture, encode and process RTI reflectance data similar to the one we are proposing in this thesis, described in detail in [Chapter: 3.2.4].

2.1 Capturing RTI data

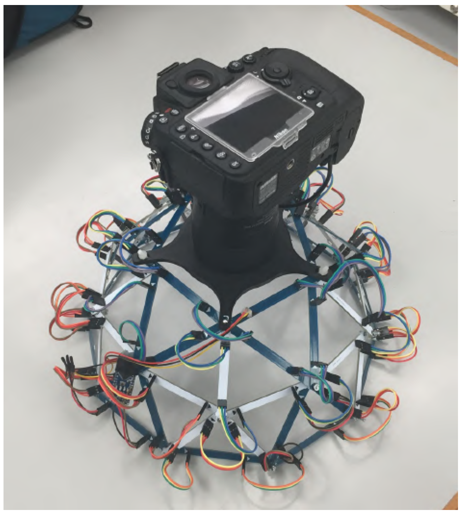
This section will elaborate some of the most common RTI acquisition methods. In [Chapter: 3.1] we replicate the methodology from [1] to acquire and extract the data necessary for RTI that is going to be discussed in [Chapter: 2.4.2].

Fixed Light Dome

The most common way to acquire RTI data is to use a fixed light dome [Figure: 2.1]. This method has the fixed light sources (from 40 to 100 LEDs) as well as the camera attached to the dome structure, and since there is no moving part, this method is quite fast. As a disadvantage, fixed light dome is much more restrictive in terms of light position, since any changes requires a reconstruction of it [48].

A lot of projects have been developed by creating an acrylic hemispherical dome of diameter 1030 mm, which is fitted with 64 flash lights and arranged in five tiers [Figure: 2.3]. During the acquisition the camera is mounted at the 'north pole' (at the highest point of the dome) and the object is placed on the horizontal baseboard [Figure: 2.2]. Thanks to the control electronics, it is possible to select or combine the desired lights and synchronised them with a digital camera. In this way, a sequence of multiple pictures of the object may be captured, each illuminated from a different direction. A geometric calibration procedure (using data of the length and orientation of shadows produced by a centrally positioned vertical pin in the baseboard) can be used in order to determine the exact coordinates of the lights [49, 50].

Figure 2.1: RTI Fixed Light Dome by Tim Zaman [35]



H-RTI

Another method is Highlight Reflectance Transformation Imaging (H-RTI), which differs from the traditional RTI acquisition approaches, because at the time of recording the photographic images, it is not required to know the locations of the lights used to construct the reflectance data [6]. The light source position is registered on each photograph and then calculated afterwards. [51]

In this method, developed by CHI around 2006, the direction of the light is figured out by spotting its specular reflection on two shiny black spheres included in the scene (using RTIBuilder). However, this approach relies on certain assumptions like constant light intensity and orthographic camera model, which can make the method unreliable in real-world situations. During the acquisition of the images, the camera is mounted on a tripod and the light source (a flash or a lamp) is manually held in position by a person moving around the object [Figure: 2.4]. For each photograph, the light source is moved to a different position and angle, always maintaining the same distance from the target, in order to create an imaginary dome around it. [48, 51]

HRTI is a flexible technique because the light source can be placed in any convenient position, but handheld acquisition is a painstaking and tedious task. In order to achieve efficiency and precision during the acquisition process, different machine setups have been developed in recent times, such as the techniques that will be presented in the following section. [48, 49]

Figure 2.2: Schematic layout of 64 lamps (white circles) on the surface of hemispherical dome, showing camera at 'north pole', the object (yellow) in the equatorial plane, and a triangular section (red) of a plane intersecting the coordinates of three lamps in Tiers 2, 3 and 5 [49]

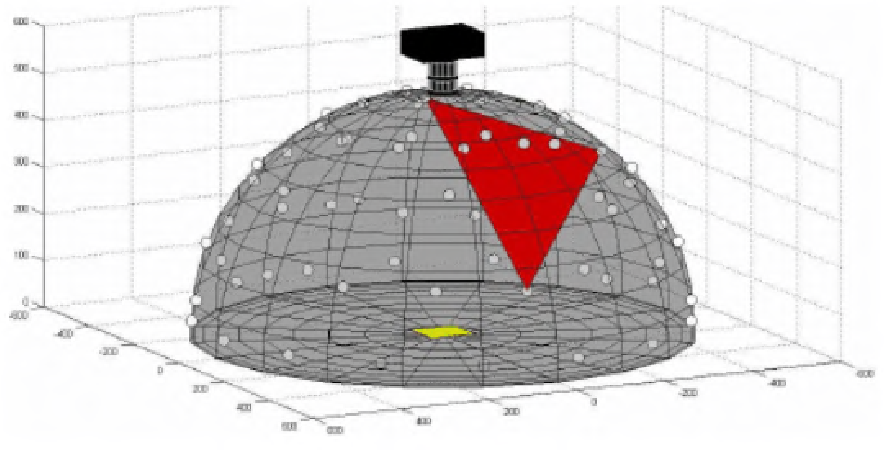


Figure 2.3: Hemispherical acrylic dome with 64 flash lamps [49]

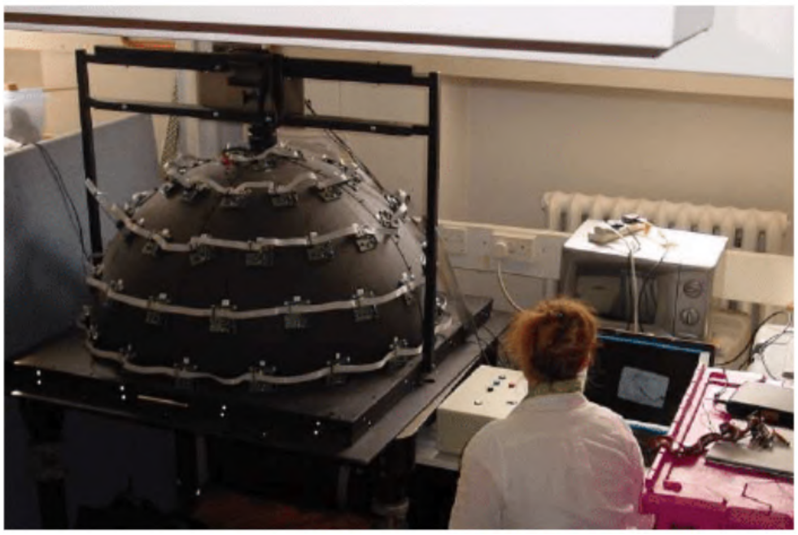
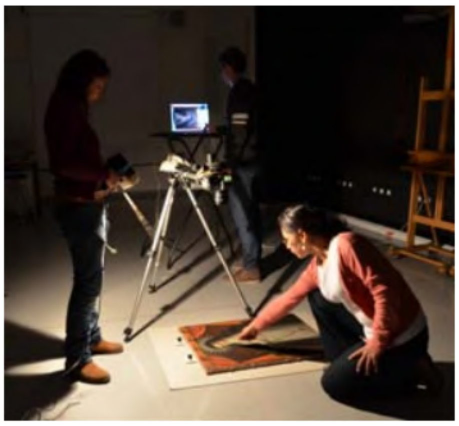


Figure 2.4: Conservators during data capture using the RTI Highlight method, where a light source is moved around the object and a new image is taken for each light angle [51]

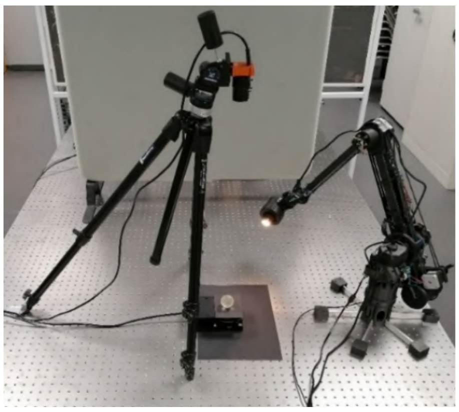


Mechanized Dome System

An alternative method for capturing RTI data involves a Mechanized Dome System, which features a servo motor that positions the light source within the dome. This setup allows users to easily select various lighting directions based on the current object without having to reconstruct the dome for each use, but at the expense of a more complex and expensive setup [48].

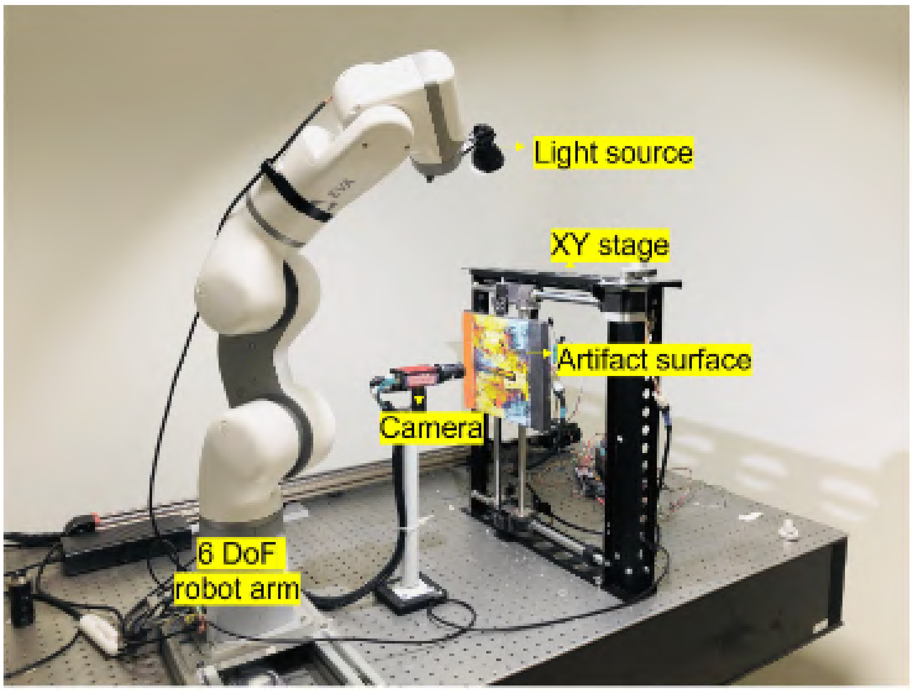
Recently, in order to capture large or translucent surfaces, robotic arm-based for RTI systems have been developed by the Norwegian Color and Visual Computing Laboratory [54]. They created a multispectral RTI system, which uses a robotic arm for positioning the light source during the acquisition. The team recognized there was a lack of RTI capture set ups of relighting models on translucent materials, so they developed this system in order to be able to acquire images of materials with different translucency. In this system, the robotic arm holds the light source and move it along a virtual half hemisphere around the object. The mechanized arm covers only half of a hemisphere, because the other half is covered by rotating the table base. The object is rotated for 180° in the horizontal plane and around the center of the virtual hemisphere. In this way, the entire azimuth range for the light source positioning is achieved [Figure: 2.5] [54].

Figure 2.5: Our RTI system consisted of a robotic arm that holds a light source, a turntable the holds the object, and a multispectral camera angle. [54]



Another impressive example is the robotic arm-based system design, which is called LightBot [53]. This system overcomes the disadvantages of the other methodologies, such as the issue of reproducibility, the limitations on the size of target's objects and difficulties on portability and speed. As it is stated in the paper, these limitations are observed in the case of free-form or dome-based systems. LightBot [Figure: 2.6] was designed to address some of these issues and, according to their knowledge, it was the first approach towards the automation of the RTI acquisition process for medium and large surfaces, which are difficult to acquire with the conventional dome systems [53].

Figure 2.6: Proposed system—LightBot [53]



2.2 Encoding reflectance data

After capturing the required images, it is necessary to encode them to be able to perform relighting in a later stage. This means that there should be a way to encode the reflectance data into a standard format. There are two commonly used software to encode (and relight) reflectance data: RTI Builder & RTI Viewer [55], that supports PTM and HSH, and Relight [56], that supports PTM, HSH and PCA/RBF, but there is no widely available software that supports methods like: Discrete Modal Decomposition (DMD) [19,20], Neural RTI [2] or On-The-Go RTI [1].

2.2.1 Polynomial Texture Maps file format

The first way to encode the reflectance data, proposed in [36] to compress PTMs, involves the creating of a color lookup table and generates a `.ptm` file. In the case of LRGB PTMs, a 256-entry lookup table, with 6 polynomial coefficients per entry, achieves quality that is close to the original PTMs without any notable increase in size. For RGB PTMs, a similar technique with a lookup table containing 18-byte entries with 12-bit indices, produces results that cannot be distinguished from the original PTMs. This method enables random access, allowing each PTM to be evaluated independently. We can also spread the polynomial coefficients into plains and use the JPEG or JPEG-LS compression algorithms to achieve better results.

2.2.2 Reflectance Transformation Imaging file format

The `.rti` file format was developed under an IMLS National Leadership grant (LG-25-06-0107-06) to the University of Southern California (USC) and Cultural Heritage Imaging (CHI) with the objective of creating a standard format that supports multiples fitting methods, like Spherical Harmonics (SH) and Hemispherical Harmonics (HSH).

2.3 Interpolation methods

After encoding the acquired images, the next step in the RTI pipeline is to use a function f to interpolate the data, that commonly takes any normalized light vector $\vec{l}$ and the encoded reflectance data p: $f(p, \vec{l}) \to y$. The following sections describe a few of the most widely used methods.

2.3.1 Polynomial Texture Maps

"Polynomial Texture Maps" [36] is the first paper that presents Reflectance Transformation Imaging.

In PTM, the core idea is to construct a polynomial regression that uses six coefficients (a_0 to a_5) to approximate the surface's reflectance behavior as a function of the angle of incident light. These coefficients are determined on a per-pixel basis and are fit to a second-degree polynomial. This polynomial equation [Eq: 2.1] incorporates the projections (l_u, l_v) of the normalized incident light vector relative to the coordinates (u,v), allowing for the detailed mapping of light across the object's surface. That formula is then used in [Eq: 2.2] and can be solved using Singular Value Decomposition (SVD) [21]. PTM provides a estimation of the overall shape and tends to smooth out fine details compared to other interpolation techniques.

$$L(l_u, l_v) = a_0 + a_1 l_u + a_2 l_v + a_3 l_u l_v + a_4 l_u^2 + a_5 l_v^2 \tag{2.1}$$

$$\begin{bmatrix} l_{u0}^2 & l_{v0}^2 & l_{u0}l_{v0} & l_{u0} & l_{v0} & 1 \\ l_{u1}^2 & l_{v1}^2 & l_{u1}l_{v1} & l_{u1} & l_{v1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{uN}^2 & l_{vN}^2 & l_{uN}l_{vN} & l_{uN} & l_{vN} & 1 \end{bmatrix} \begin{bmatrix} a_5 \\ a_4 \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \vdots \\ L_N \end{bmatrix} \tag{2.2}$$

2.3.2 Hemispherical harmonics

Hemispherical Harmonics (HSH) [17, 18], derived from Spherical Harmonics (SH) functions, represent an evolution of PTMs. The core concept behind HSH is to project the data onto a more appropriate set of hemispherical harmonics basis functions, which more closely resemble the shape of the reflectance field. The HSH functions, denoted as H_l^m, are derived from Legendre Polynomials [Eq: 2.3] and are only defined for the upper hemisphere [Eq: 2.4], where K_l^m are the hemispherical normalization factors [Eq: 2.5]. Now, the surface reflection function can be broken down into a series of Hemispherical Harmonics (HSH) of varying order 1 and degree m [Eq: 2.6].

$$\tilde{P}_l^m(\cos\theta) = P_l^m(2\cos\theta - 1) \quad \text{and} \quad \theta \in \left[0, \frac{\pi}{2}\right] \tag{2.3}$$

$$H_l^m(\theta,\phi) = \begin{cases} \sqrt{2K_l^m}\cos(m\phi)\hat{P}_l^m(\cos\theta) & \text{if } m > 0 \\ \sqrt{2K_l^m}\sin(-m\phi)\hat{P}_l^{-m}(\cos\theta) & \text{if } m < 0 \\ K_l^0\hat{P}_l^0(\cos\theta) & \text{if } m = 0 \end{cases} \tag{2.4}$$

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)}{2\pi(l+|m|)}} \tag{2.5}$$

$$C_l^m(\theta_v,\phi_v) = \int_0^{2\pi}\int_0^{\frac{\pi}{2}} f(\theta_v,\phi_v,\theta_i,\phi_i)H_l^m(\theta_i,\phi_i)\sin\theta_i d\theta_i d\phi_i \tag{2.6}$$

2.3.3 Radial Basis Functions

Another popular method for interpolating the RTI data is using Radial Basis Function (RBF) [14]. RBF, originally introduced by [13] for solving partial differential equations, is a general mathematical technique applicable in various fields beyond RTI, such as neural networks [15] [16], pattern recognition, data visualization, medical applications, surface reconstruction, etc.

RBFs can be used for function interpolation. Given a set of points, they can create a function that fits the points at their locations. RBF interpolation is a mesh-free method, meaning it does not require a grid to be defined. This makes it very flexible for multi-dimensional data interpolation. A commonly used form for RBF interpolation is:

$$s(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \phi(\|\mathbf{x} - \mathbf{x}_i\|)$$

Where:

- $s(\mathbf{x})$ is the interpolant.
- ϕ is the radial basis function.
- λ_i are coefficients to be determined.
- $\mathbf{x}_i$ are the centers (the given points).
- N is the number of centers.

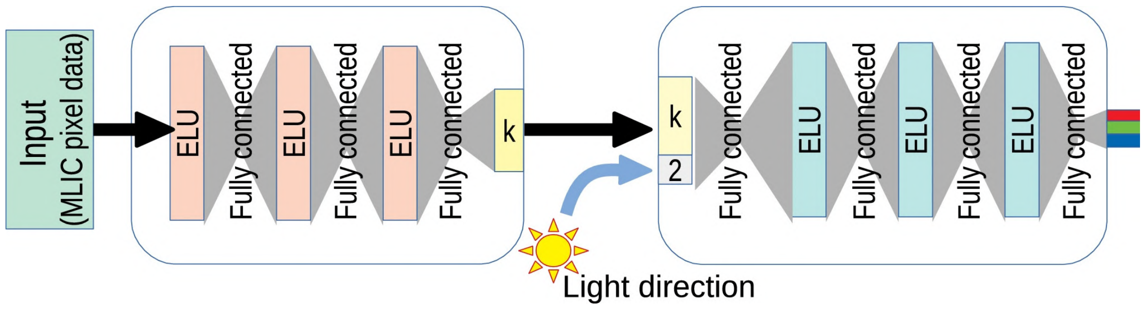
2.4 Neural networks based methods

These approaches utilize neural networks to facilitate RTI by, for example, leveraging an encoder/decoder architecture. They can efficiently compress reflectance data [23] while preserving high-quality details. Additionally, they are capable of accurately approximating the light transport function, essential for the relighting. Two methods will be presented.

2.4.1 Neural RTI

In "Neural reflectance transformation imaging" [2] the authors propose to use a simple Neural network with an autoencoder architecture [Figure: 2.7] to compress the huge amount of information, which RTI requires into a much smaller format that better preserve the original information compared to traditional methods. Autoencoders [25] are neural networks capable of learning to compress the input and later to decompress it as close as possible to the original, this type of neural network has been proven useful for tasks like image compression [26] and denoising [27].

Figure 2.7: Neural RTI model architecture



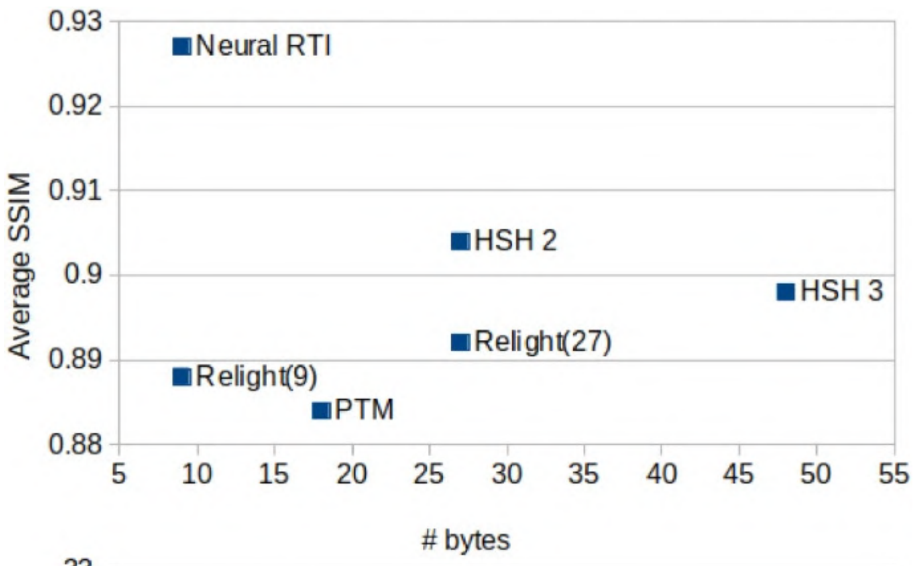
The described model is trained using a collection of multi-light images, with the pixel data of these images and their corresponding light directions as input. The training aims to minimize the difference between the network's predicted pixel values and the ground truth pixel values for a set of known light directions.

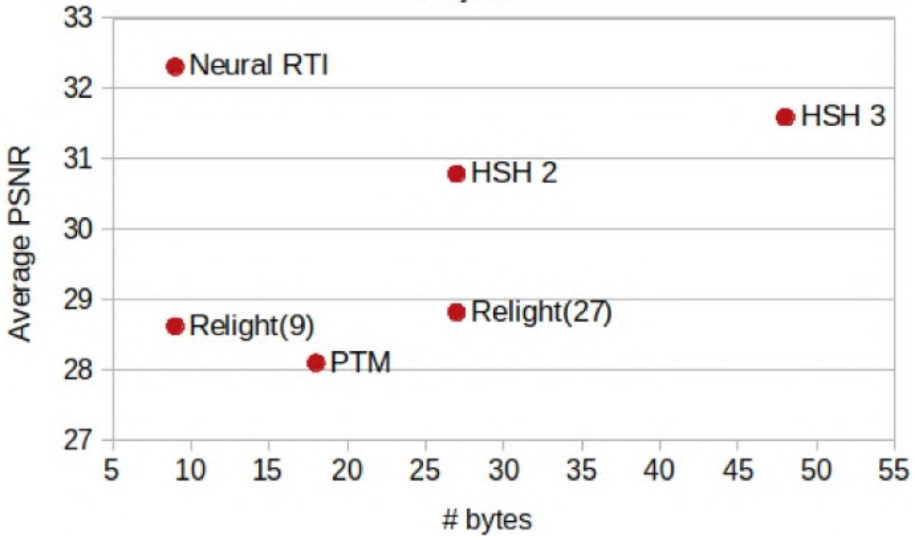
The network is designed to create a storable relightable image composed of a compact pixel code along with the coefficients of the decoder network, which are unique for each pixel location, and then, given any light direction, it generates relighted images.

The main metrics used to evaluate the model were Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity index (SSIM), both metrics will be discussed in [Chapter: 3.3.2] and [Chapter: 3.3.3] respectively.

Neural RTI has proven to be able to provide better results than current state-of-the-art methods [Figure: 2.8] because it requires less space to store the data, making it even possible to run it on the web using tensorflow.js [39]. In the same paper the authors presented the SynthRTI dataset discussed in [Chapter: 4.1].

Figure 2.8: Neural RTI average SSIM and PSNR on SynthRTI



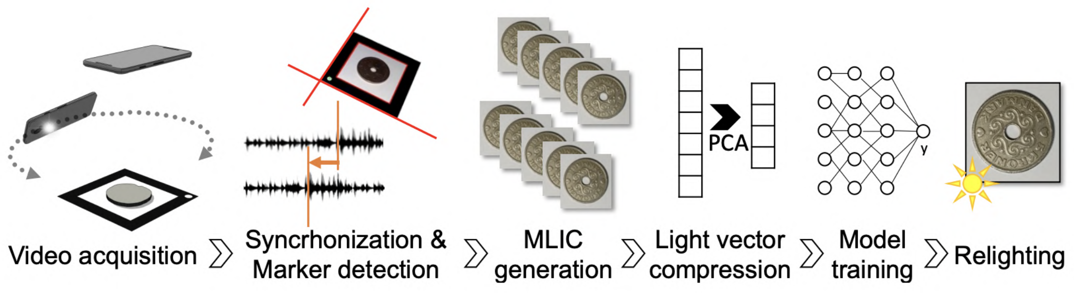


2.4.2 PCA model

In the work titled ”On-the-go Reflectance Transformation Imaging” [1] the authors introduce a novel approach for capturing reflectance data using two smartphones. Additionally, they present a new neural relighting model capable of reconstructing the appearance of objects under any lighting direction. This model efficiently reconstructs object appearance from reflectance distribution data compressed utilizing Principal Components Analysis (PCA).

In this thesis, we replicated the methodology outlined in the referenced paper, starting with data extraction from smartphone videos, as detailed in [Chapter: 3.1], through the compression and interpolation of reflectance data utilizing PCA and their neural model.

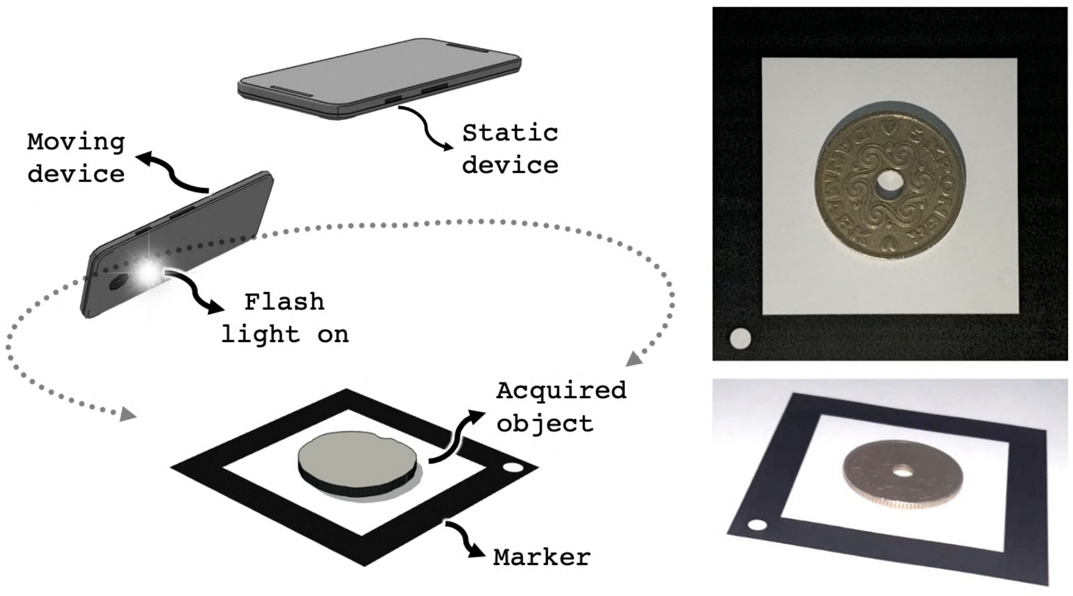
Figure 2.9: RTI acquisition and relighting pipeline [1]



The entire pipeline described in the paper, that can be visualized in [Figure: 2.9], is divided in several steps, from the video acquisition and data extraction (using audio synchronization and marker detection) to the creation of the Multi-Light Image Collection, then to the PCA light vector compression, model training and finally relighting.

The initial steps of the pipeline are dedicated to the reflectance data extractions for creating the MLIC. It starts with the video acquisition shown in [Figure: 2.10]. The video is captured using two ordinary smartphones, by placing one, denoted *static*, above the object with the camera towards it and the other one, denoted *moving*, that will record the video with the flashlight turned on while moving around the object. The purpose of this method is to use the *moving* smartphone flashlight instead of the fixed light domes LEDs and uses the frames from the *static* smartphone as the captured images. The two videos from the smartphones are then synchronized using the audio track and the marker (with the object placed inside it) serves a different purpose for each camera. For the *static* camera the marker it's used to detect the corners to crop the images ($W \times H$ pixels) that will compose the final MLIC. It's worth noting that the images are captured in greyscale first and then re-colored later on, but in this thesis we only reproduced the greyscale version. For the *moving* camera the marker is used to compute the Homography H, factorising it as [Eq: 2.7] and then compute the light direction L [Eq: 2.8]

Figure 2.10: Left: RTI acquisition methodology, Right: example of images obtained with the two smartphones [1]



$$K^{-1}H = \alpha \begin{bmatrix} | & | & | \\ r_1 & r_2 & t \\ | & | & | \end{bmatrix} \tag{2.7}$$

$$\alpha \approx 2/(||r_1|| + ||r_2||)$$

$$L = t/||t|| \tag{2.8}$$

Where:

- K is the intrinsic camera matrix
- r_1, r_2 are the first two columns of the rotation matrix R
- α is a non-zero unknown scale factor that can be approximated since R must be orthonormal
- t is a vector that represents the light position
- L is final light direction

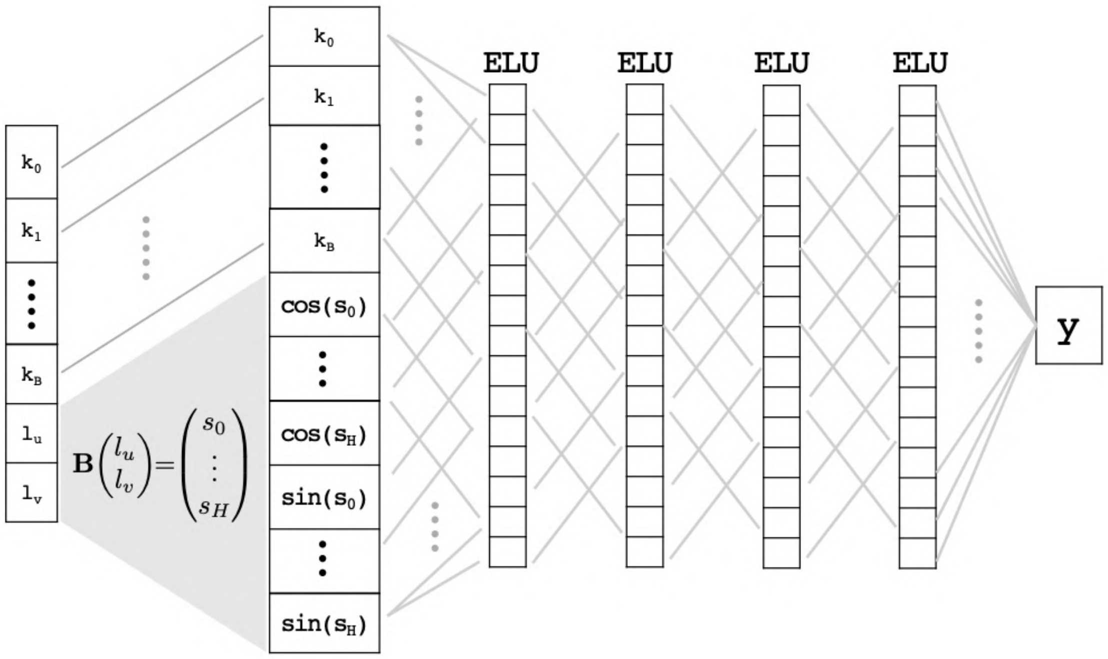
After the initial steps a MLIC with the corresponding light directions is obtained and it's possible to compress the light direction vector via Principal Components Analysis. To do that they defined B (in their case 8) orthogonal bases for a total compressed light vector K of $W \times H \times B$ values.

The next step in the pipeline is training the network Z. The proposed model is a small Multilayer Perceptron (MLP) [Figure: 2.11] that takes in input the light compressed vector k for the desired pixel coordinates and the light vector $\vec{l} = (l_u, l_v)$ of size B. The model uses positional encoding [30] on the light vector to increase performances, this means that the vector is projected into a H dimensional Fourier space as explained in [Chapter: 1.3], making the input of the model $B + 2H$ in size that looks like [Eq: 2.9].

$$I = (k_0, \ldots, k_B, \cos(s_0), \ldots, \cos(s_H), \sin(s_0), \ldots, \sin(s_H)) \tag{2.9}$$

Where $S = B * \vec{l}$, and B is a randomly sampled Gaussian distribution $N(0, \sigma^2)$.

Figure 2.11: PCA model architecture [1]


k_0
k_1
k_B
l_u
l_v
$\mathbf{B}\begin{pmatrix} l_u \\ l_v \end{pmatrix} = \begin{pmatrix} s_0 \\ \vdots \\ s_H \end{pmatrix}$
k_0
k_1
k_B
cos(s_0)
cos(s_H)
sin(s_0)
sin(s_H)
ELU
ELU
ELU
ELU
y

The last step, the relighting, can be carried out with [Eq: 2.10]

$$f(p, \vec{l}) = (\mathcal{Z}(k_p, \vec{l}), \bar{U}(p), \bar{V}(p)) \tag{2.10}$$

Chapter 3

Proposed Methodology

This chapter will describe our methodology to obtain the results available in [Chapter: 4] and to reproduce the results from [1]. We started by extracting the data using smartphone videos [Chapter: 3.1] and then proceed by interpolating the data with several methods [3.2]. After reproducing the results, we started implementing a new interpolation method using our own neural network with an INR [Chapter: 3.2.4].

3.1 Data Extraction

This section will explain how the data of the coins dataset was extracted, from how the images were captured to our methodology for calculating the light direction from them.

3.1.1 Video Creation

For each coin, two videos were created:

- A video from a static camera above the object

- A video from a camera that was moving around the object with the flashlight turned on

Those videos were recorded at the same time. For reproducing the acquisition process, it is necessary to follow these steps:

- Place a known planar square marker on a flat surface, and place the target object on top of the marker.
- Place the first smartphone above the object with the flashlight turned off, and start the camera to record a video sequence.
- Set the second smartphone to have both the camera and flashlight turned on, and move it around the object to shine light on it from different angles. Be sure to keep the marker visible in the frames.

3.1.2 Video Syncing

As stated earlier, each coin corresponds to two videos, but synchronization does not occur inherently between them. This causes a problem because the frames from the moving camera and the static camera needs to be aligned in order for the algorithm to be able to correctly extract the light direction of each frame.

To address this issue, we have to solve the following problems:

- There is a delay between the start of the first and the second video.
- The two videos have two different FPS values.

Solving the video delay

The best way to find the delay between the two videos is to look at the audio track of the videos directly and align it. This is possible because the two videos were recorded at the same time, so the audio track between the two should be very similar. Since we could control the video creation, we intentionally snapped our fingers during each recording session to make the audio alignment easier. To align the audio tracks we used an open source tool called Audacity [37], a popular audio editing software, to find the timestamp of the snapping fingers in both videos, and easily calculate the delay (d) between the twos, using [Eq: 3.1]

$$d = t_s - t_m \tag{3.1}$$

Where:

- d is the total delay.
- t_s is the timestamp of the static video.
- t_m is the timestamp of the moving video.

After calculating d we used another open source tool for video editing called FFmpeg [38] to create a new video of the moving camera that had a blank offset at the start using the `input` function by setting the `itsoffset` attribute equals to d. Now we have two new videos that have the exact same starting point.

Solving the fps difference

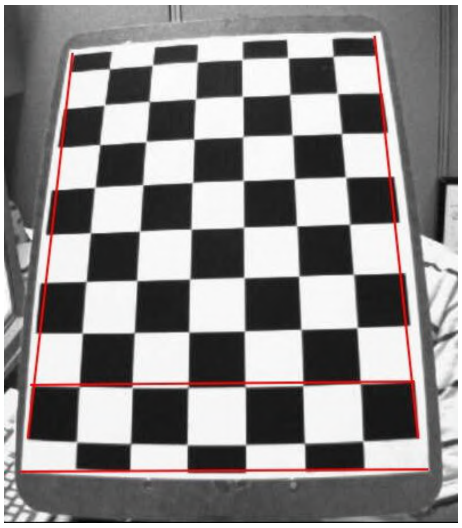
If the two videos have different fps values, the videos will get little by little out of sync during the analysis. The reason for this is that our code analyzes every frame of each video, so if one of the two has more frames than the other one, the twos will not be synced anymore. To solve this we can, once again use FFmpeg [38] to bridge the gap and generate two new videos with the same fps values, using the `filter` function with an equal `fps` attribute value. Now we have 2 new videos that have the exact same fps value.

3.1.3 Camera Calibration

Pinhole cameras can often cause considerable image distortions, typically in two primary forms: radial and tangential distortion.

Radial distortion results in straight lines appearing curved in the image. This distortion effect amplifies as the distance from the image center increases. For instance, consider the image shown below [Figure: 3.1], where a chess board's two edges are highlighted with red lines. It becomes evident that the chess board's border deviates from a straight path and does not align with the red line. All anticipated straight lines are visibly distorted, curving outward.

Figure 3.1: Image distortion problem



3.1.4 Finding the camera intrinsic matrix

In order to solve the distortion problem we first need to find the intrinsic matrix of the camera. To do that we used the `findChessboardCorners` function and the `calibrateCamera` function from OpenCV [40]. In order to use this functions we need to have some pictures of a standard chessboard that can be generated with a dedicated script [43]. Since we are using two smartphones, the camera properties while recording a video were different than while taking pictures, in order to have the correct ones we had to obtain the images of the chessboard directly from a video of the chessboard. Another point to consider is that only a few frames are required for effective camera calibration, so if we took a short video of the chessboard, we can skip most of the images by setting the current frame of the video to $f_c + f_s$, where each iteration and making sure that [Eq: 3.2] is always respected to prevent overflowing. All of this translates to roughly the following code:

$$f_c + f_s \leq n \tag{3.2}$$

Where:

- f_c is the current frame
- f_s is the number of frames to skip
- n is the total number of frames in the video

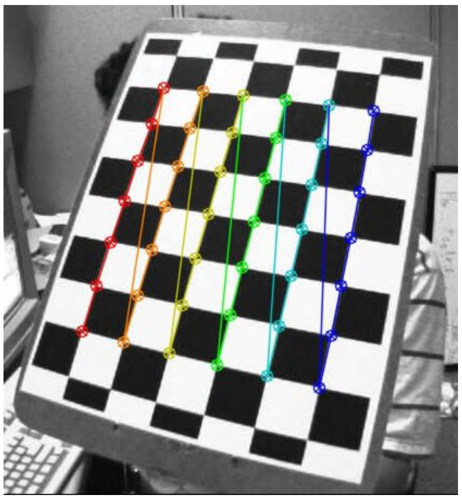
Now that we saw how to get frames from the calibration video, we can use the `findChessboardCorners` and `calibrateCamera` functions to find:

- r, the RMS of the process
- K, the intrinsic parameters matrix of the camera
- d, the distortion parameter
- $\vec{r}$, the rotation vectors
- $\vec{t}$, the translation vectors

Of these parameters we mostly care about r that, for a good calibration result, should be between 0 and 1 (ours was ≈ 0.6 for the static camera and ≈ 0.7 for the moving camera), K and d, that are used in the `undistort` function later on.

In order to visually see if the calibration is finding corners during the process, we can use the `drawChessboardCorners` function from OpenCV [40] [Figure 3.2].

Figure 3.2: Camera calibration process



Since the camera intrinsic matrix do not change unless you use a different camera, the camera calibration is a step that can be performed once for each camera. This means that we need to be able to save the resulting values in a file. Thankfully OpenCV [40] provides a dedicated class to do this called `FileStorage` where we can save: r, K and d.

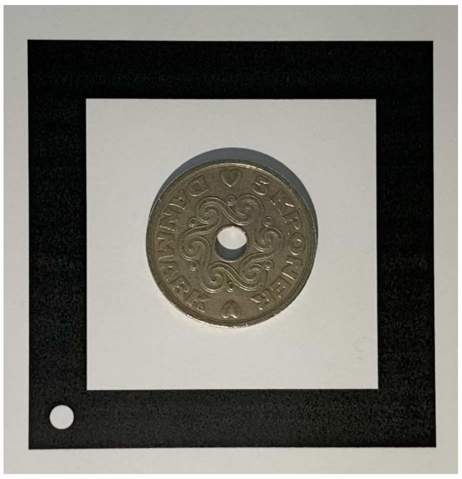
Un-distorting the image

Now that we have the camera intrinsic matrix we can use them to un-distort the image of the videos. To do that we used the `undistort` function from OpenCV [40], with the distorted frame, d and K.

3.1.5 Marker Detection

For each frame of the videos that we extract we need to find the black rectangular marker in it [Figure: 3.3].

Figure 3.3: Image of a coin inside the marker



There are different ways of doing this, for example you could use a combination of image processing techniques, like blur, erosion and dilation with the Canny edge detector [45], or any thresholding algorithm like Otsu [46] and Adaptive thresholding [47](we chose this one). After processing the image we can use the `findContours` and filter only polygons with four edges (rectangles).

Now that we have our four points of the rectangles we need to sort the corners in order to have consistency (the first point must be the top-left, the second the top-right, . . .). We can do this by calculating for each corner the point rotation angle [Eq: 3.3] and sorting them using it.

$$\alpha = \arctan(\frac{p_x - c_x}{p_y - c_y}) \tag{3.3}$$

Now we have a problem: the `findContours` function should find two big rectangles, the outer one and the inner one, but the camera rotates during the video, changing the corners order. To maintain the same order of corners we need to look for the white dot in one of the corners of the marker. After finding it we can set that corner as the first one, and the rest will stay consistent throughout the analysis.

In order to simply find the white dot we can first find the homography of the marker, with that we can check the pixel area where the white dot should be and rotate the corners until we find it.

3.1.6 Camera Pose Estimation

Since we can approximate the light direction from the point of view of the static camera as the position of the moving camera itself, we need to find a way to estimate the moving camera position in the space.

We can calculate the moving camera position using the formula [Eq: 3.4].

$$c_p = -R^T * \vec{t} \tag{3.4}$$

Where:

- c_p is the camera position
- R is the rotation matrix in the marker's coordinate system
- $\vec{t}$ is the translation vector in the marker's coordinate system

We obtained the R and $\vec{t}$ values with the `solvePnP` function from OpenCV [40]. Now we can normalize it so that the $\vec{l}$ has a magnitude of 1 [Eq: 3.5]

$$\vec{l} = \frac{c_p - a}{||c_p - a||} \tag{3.5}$$

Where:

- $\vec{l}$ is the light vector
- c_p is the camera position
- a is the center position of the marker

$\vec{l}$ will now contain normalized x and y values between -1 and +1, and we can display it in a 2D plot.

3.2 Data Interpolation

Data interpolation refers to the method of generating an image based on the light outputs captured during data extraction and any light direction. We implemented two distinct approaches to interpolation. The first is **precomputed** interpolation, where we store the resulting image for every potential light direction within a predefined confined dome. This approach facilitates quick and efficient interactive relighting [Chapter: 3.4]. The second approach is **real-time** interpolation, where each image is calculated on-the-fly during the relighting process. This method avoids the need for pre-storing images, thus saving both space and time. The interpolation functions we have developed include:

3.2.1 LinearRBF

We implemented a Linear Radial Basis Function described in [Chapter: 2.3.3] using the SciPy library [41].

3.2.2 Polynomial Texture Maps

The second function we implemented is Polynomial Texture Maps (PTM), the first RTI method outlined in [Chapter: 2.3.1], without using any external libraries except NumPy [42], that provided useful functions to handle matrices and Singular Value Decomposition (SVD).

3.2.3 PCA Model

We replicated the model described in the research paper titled 'On-the-go Reflectance Transformation Imaging with Ordinary Smartphones' [1], described in [Chapter: 2.4.2], that uses as relighting function a machine learning model that takes in input a extremely compact reflectance data compressed via Principal Components Analysis (PCA).

3.2.4 Implicit Neural Model

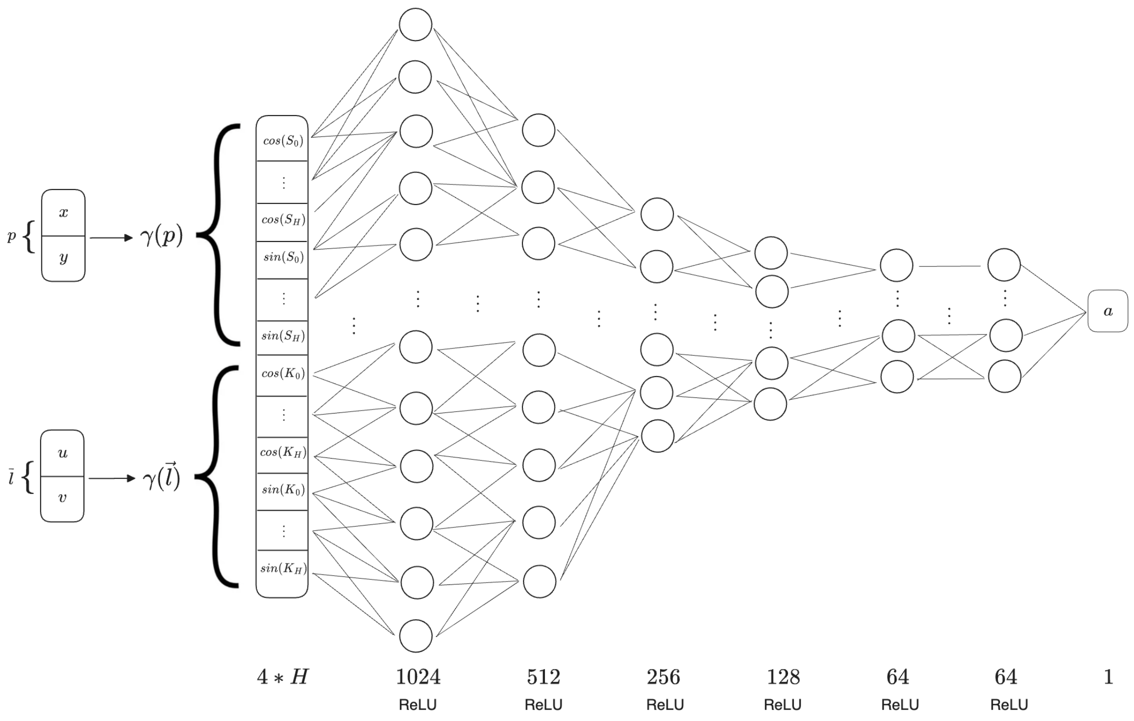
The final interpolation method we are going to cover is the Implicit Neural Model interpolation. This approach uses a machine learning model that we are now proposing, denoted as $Z(x, y, \vec{l}) \rightarrow a$, which utilizes an implicit neural representation [Chapter: 1.3]. The model interpolates the intensity a of a pixel at coordinates $p = (x, y)$ for any specified light vector $\vec{l} = (l_u, l_v)$, where $l_u, l_v \in [-1, \ldots, +1]$.

The underlying model is a Multilayer Perceptron (MLP) consisting of seven linear layers interlaced with seven ReLU activation functions [Figure: 3.4]. The input of the model can be represented as $I = [x, y, u, v]$, where (x, y) are the pixel coordinates of p and (u, v) are the light direction components of $\vec{l}$. Both sets of inputs are independently projected into a higher-dimensional Fourier space with random frequencies, to enhance the network's performance [Chapter: 3.2.4], this method is inspired by [30]. This means that the model Z initial layer is non-trainable and the pixel coordinates and light direction are projected into it [Chapter: 3.2.4].

Projecting the model input into Fourier spaces

As previously said, the first layer of the network Z is a non-trainable projection of the pixel coordinates (x, y) and light direction $\vec{l} = (l_u, l_v)$ into two distinct higher-dimensional Fourier space with random frequencies. More specifically we want to

Figure 3.4: MLP network architecture composed by 7 fully-connected layers with ReLU activation



map both our pixel coordinates and light direction to [Eq: 3.6], and concatenating the results [Eq: 3.7]

$$\gamma(A) = [\cos(2\pi b_1^T A), \dots, \cos(2\pi b_H^T A), \sin(2\pi b_1^T A), \dots, \sin(2\pi b_H^T A)]^T \tag{3.6}$$

$$\gamma(I) = \gamma(p) \frown \gamma(\vec{l}) \tag{3.7}$$

Where b is a randomly sampled gaussian matrix $N = (0, \sigma^2)$ of size H (in our case 12). The resulting network input I is a $4 * H$ dimensional vector created by concatenating the projection of the coordinates and the projection of the light direction.

To create the gaussian matrices in Python we can use the `torch.normal()` function from PyTorch [44].

We used 0 as mean value, and turned the standard deviation into a hyperparameter of the model Z, giving a final value of 3.0 for the gaussian matrix of the coordinates and 0.6 for the gaussian matrix of the light directions.

Network training

Our model was trained with a dataset that mapped every input I to the resulting reflectance intensity for that pixel. This means that for every pixel of every image in the train set we created a record $[x, y, u, v] \rightarrow a$ and added it to our dataset. The total number of data sample in the dataset is equal to $W * H * N$, with W and H representing the width and height of the images, and N represents the total number of light directions available in the train set. After acquiring the data we shuffled it randomly to prevent any order bias on the model and improve the overall performance on unseen data. We trained the model using the Adam optimizer [28] with a learning rate $\alpha = 0.01$ and a StepLR scheduler from PyTorch [44] with $\gamma = 0.1$ and $step\_size = 15$. Regarding the loss function we used an L1Loss that implements the Mean Absolute Error (MAE) loss [Chapter: 3.3.1]

Using a perceptual loss

While we were trying to implement the Implicit Neural Model, our model was very accurate in datapoints that were part of the dataset, but very inaccurate everywhere else. Out hypothesis to fix this, was to use a perceptual loss [29] instead of the L1Loss directly. To obtain the perceptual loss between two images you must use a model like Vgg16, that is implemented in PyTorch [44], on both images to create a set of features that can now be inserted in the L1Loss instead of the two images. To test that our hypothesis was correct, we wrote a function to obtain N points between

Figure 3.5: Linear growth

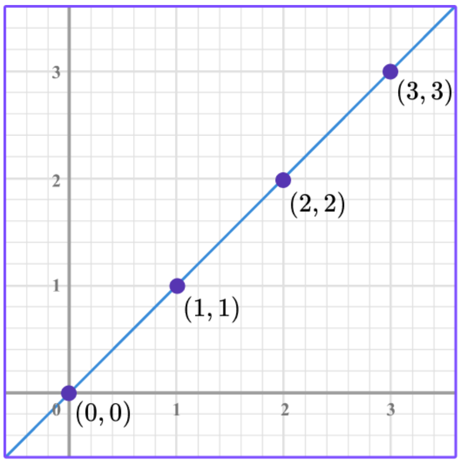
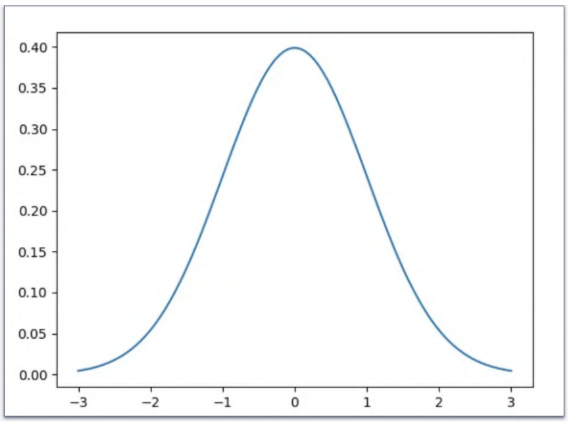


Figure 3.6: Gaussian distribution



two other, and calculated the perceptual loss between the image interpolated by our model in the first point and every image interpolated in the other N points. By generating a graph of the resulting losses, we were able to determine the effectiveness of utilizing perceptual loss. Specifically, if the graph resembled a normal distribution, similar to [Figure: 3.6], the loss would be ideal. However, the actual graph showed a linear growth pattern, like the one in [Figure: 3.5], indicating that our hypothesis was incorrect.

3.3 Data Analysis

We implemented several standard algorithm to calculate some analytical values to confront our different interpolation methods in a scientifically accurate way, the main problem is that confronting images analytically is not an easy problem to solve, so no one algorithm is better than others, in this chapter we'll expand a bit on each algorithm.

3.3.1 L1 (Mean Absolute Error)

L1 loss, also known as mean absolute error (MAE), is a popular loss function used in machine learning for regression problems that is available out-of-the-box using PyTorch [44]. It generally measures the absolute differences between predicted values and actual values, providing an assessment of prediction accuracy. It can be defined mathematically with [Eq: 3.8], where x is the input, y is the target, N is the batch size and μ is the mean function (in our case).

$$\ell(x, y) = \mu(L) = \{l_1, \dots, l_N\}^T, \quad l_n = |x_n - y_n| \tag{3.8}$$

3.3.2 PSNR

Peak Signal-to-Noise Ratio (PSNR) (here denoted with ψ) is a measure of the quality of reconstruction of lossy transformation processes, so it's a value suited to be used in benchmarks for signals or images. It can be defined using the Mean Square Error (MSE) (here denoted with γ) [Eq: 3.9], with [Eq: 3.10].

$$\gamma = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} ||I(i,j) - K(i,j)||^2 \tag{3.9}$$

$$\psi = 20 \cdot \log_{10} \left(\frac{M_I}{\sqrt{\gamma}} \right) \tag{3.10}$$

Where:

- M and N are the dimensions of the image
- I is the original image
- M_I is the maximum possible pixel value of the image, for example 255
- K is the altered image that we want to evaluate

The MSE measures the average of the squares of the errors between the original and altered images. A higher PSNR generally indicates that the reconstruction is of higher quality. It's worth noting that in some contexts an image with a higher PSNR may not actually look better to human observers than one with a lower PSNR. Nevertheless, it remains a commonly used standard for objective measurement of image quality.

3.3.3 SSIM

Structural Similarity index (SSIM) is a method for measuring the similarity between two images designed to improve on traditional methods like PSNR that are sometimes inconsistent with human eye perception. The SSIM index provides a value between -1 and 1, where 1 is only reachable in the case of two identical sets of data and therefore indicates perfect structural similarity and value of 0 indicates no structural similarity.

The basic SSIM index (here denoted by ϕ) is calculated on various windows of an image of measure x and y of common size $N \times N$ like [Eq: 3.11].

$$\phi(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.11}$$

Where:

- μ_x is the pixel sample mean of x
- μ_y is the pixel sample mean of y
- σ_x^2 is the variance of x
- σ_y^2 is the variance of y
- σ_{xy} is the covariance of x and y

- $c_1 = (0.01 * L)^2, c_2 = (0.03 * L)^2$ are constants to stabilize the division obtained with the dynamic range of the pixel values L

3.4 Interactive Relighting

In order to view the output of the interpolated data, we need a program that allows us to do **Interactive relighting**, meaning a program that for any light direction in input, displays the interpolated output image.

Our solution supports two different versions of interactive relighting: Real-time relighting and Pre-computed relighting. Our Real-time solution computes the output of the interpolation function on the fly, not needing any additional steps after extracting the data. The downside of the Real-time approach is that some interpolation functions in particular can be computationally expensive, meaning that from an average device it's practically impossible to have a nice user experience. Pre-computed relighting on the other hand, alleviate the computation needed during relighting by adding an additional pre-computational step after analysis. This step will generate every possible image for every possible light direction and store it in an archive. The archive can be pretty large depending on the size of the images and on how many possible light direction the interactive relighting programs supports. This means that during the relighting, no interpolation function is running because it will just display the images from the pre-computed archive.

3.4.1 Obtaining the light direction

In our implementation the light direction is inputted through a Graphical User Interface (GUI). This allows us to input the light direction in a fast and intuitive way. Our implementation leverages OpenCV [40] to create an empty frame that will create a line between the selected point in the frame. The line is created from the center to the selected point [Eq: 3.12].

$$\begin{bmatrix} x_1 = \frac{1}{2} \cdot w_s \cdot \rho \\ \\ y_1 = \frac{1}{2} \cdot w_s \cdot \rho \end{bmatrix} \rightarrow \begin{bmatrix} x_2 = l_x \cdot \rho \\ \\ y_2 = l_y \cdot \rho \end{bmatrix} \tag{3.12}$$

Where:

- w_s is the window size
- ρ is the scaling factor

After creating the frame we used OpenCV's `setMouseCallback()` to attach a event handler to the frame interactive on click. The function changed the current light direction to a new one based on the mouse location and constrained the possible coordinates to a circle of radius 1 from the center.

Chapter 4

Experimental Results

In this chapter, we conducted experiments with our model on the SynthRTI dataset [Chapter: 4.1] as well as an internal dataset that has not been made public yet. The process of tuning our hyperparameters will be detailed in [Chapter: 4.2], and in [Chapter: 4.3] we are going to compare our solution with other state-of-the-art solutions.

4.1 SynthRTI dataset

To test our model we used the Synthetic dataset created by Dulecha, T.G., Fanni, F.A., Ponchio, F. et al [2]. This dataset provides multi light images collection of an object created with Blender Cycles rendering engine [32] and it provides a benchmark that better represents real-world data from RTI applications than other dataset that could be used for testing relighting quality, like DiLiGent [12]. The dataset is divided into two subsets:

- SingleMaterial: three different objects with eight different materials each ($8 * 3 * n_l$ total images)

- MultiMaterial: three different objects with nine different materials combined each ($9 * 3 * n_l$ total images)

In this context, n_l represents the number of lights available in the dataset. This number varies depending on the lighting setup: the classical dome light set (which includes 49 lights) or the test light set (comprising of 20 lights). The purpose of this division is to train the model using the classical dome light set and then assessing the model's performance using the test light set.

The three objects present in the dataset are:

- 3D scan of an oil on canvas painting by W. Turner, performed by R.M. Navarro and found on SketchFab [33] (a nearly flat surface)
- Scan of a cuneiform tablet from Colgate University
- Scan of relief in marble "The dance of the Muses on Helicon" by G. C. Freund, digitized by G. Marchal.

The dome's lighting arrangement consists of 49 lights placed in concentric circles at five different elevation levels: 10, 30, 50, 70, and 90 degrees. In [Figure: 4.1], this setup is depicted on a 2D plane, where the main set of 49 dome lights are represented as blue dots, and an additional set of 20 test lights are shown as red dots.

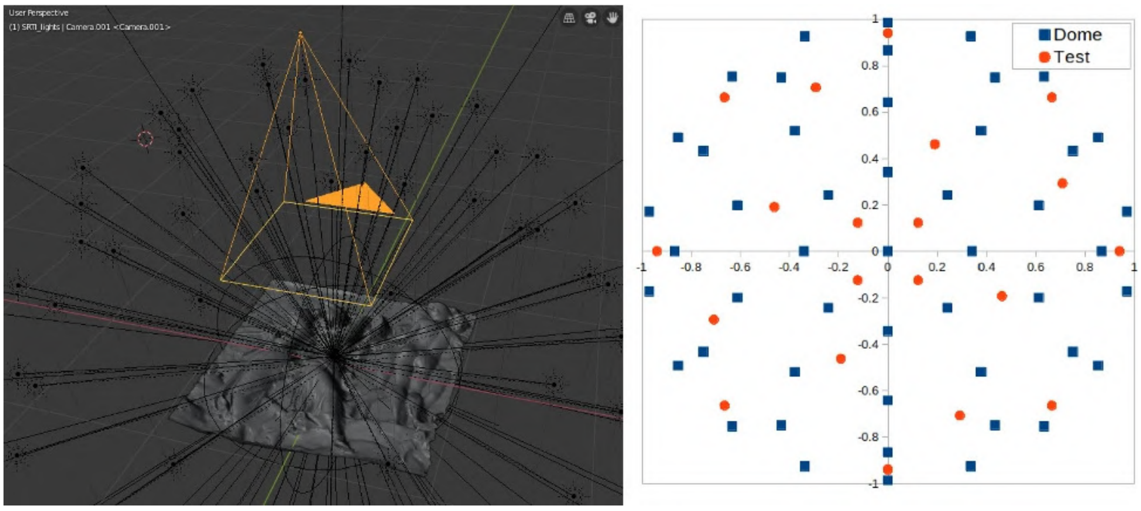
In our scenario, our primary focus is on utilizing the dataset for Reflectance Transformation Imaging (RTI). However, it's worth noting that the dataset could also be effectively applied in Photometric Stereo methods [34].

The dataset is publicly available on Github [31].

4.2 Tuning the model

In this section we'll show how we have chosen our hyperparamentes for the Implicit Neural model.

Figure 4.1: Blender viewport and light direction dome [2]



4.2.1 Finding σ_l value

In [Table: 4.2][Graphs: 4.2] we can see the analytical results we obtained while varying the σ_l in our implicit model. And in [Figure: 4.4] we can see the corresponding model outputs, where each row represents a different σ_l (from 0.45 to 0.8) with 3 different output. The chosen $\vec{l}$ for the output images are: $(0.157, 0.484)$, $(0.161, 0.884)$ and $(0.726, 0.615)$, knowing that $(l_u, l_v) \in [-1, \ldots, +1]$.

As you can see the values we got are very similar, but we can see that we are getting better looking images around $\sigma_l = 0.55$, so that's what we have chosen.

4.2.2 Finding σ_p value

We applied the same methodology as [Chapter: 4.2.1] to find the right σ_p value. We can see the analytical results in [Table: 4.1][Graphs: 4.3] and output images in [Figure: 4.5].

$\sigma_\mathbf{p}$	**SSIM**	**PSNR**	**L1**
1.0	0.8692	26.3127	5.5796
1.5	0.8875	27.4911	4.8785
2.0	0.9347	30.8701	3.3406
2.5	0.8842	28.2619	4.8491
3.0	**0.9432**	**32.0739**	**3.0084**
3.5	0.7812	25.3905	7.2371
4.0	0.9239	31.1403	3.5118
4.5	0.8524	27.7357	5.3937
5.0	0.9028	30.1158	4.0419
5.5	0.8290	27.4266	5.7920

Table 4.1: Metric values for different σ_p values

$\sigma_\mathbf{l}$	**SSIM**	**PSNR**	**L1**
0.35	0.8713	28.0377	5.0719
0.4	0.8697	27.9668	5.1482
0.45	0.8945	28.8153	4.4759
0.5	0.9311	31.0283	3.4131
0.55	**0.9432**	**32.0739**	**3.0084**
0.6	0.8563	26.9683	5.5924
0.65	0.8668	27.8119	5.2548
0.7	0.8696	27.6481	5.2379
0.75	0.8646	27.7329	5.2793
0.8	0.9296	30.6438	3.4950

Table 4.2: Metric values for different $\sigma_{\vec{l}}$ values

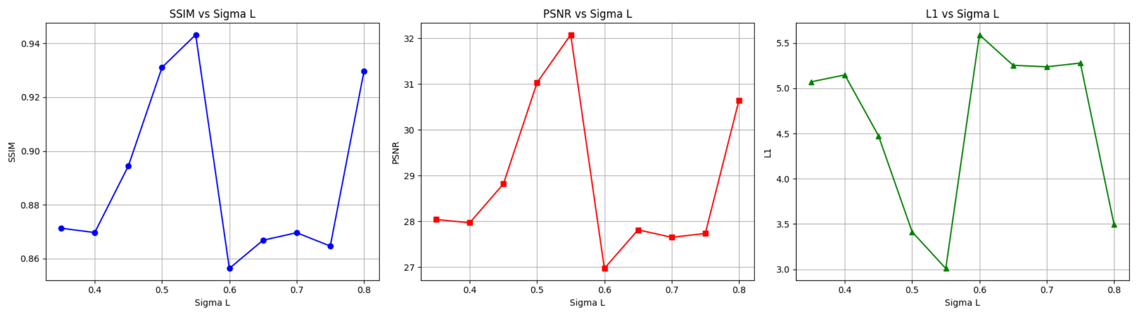


Figure 4.2: Graphs of the metric values for different $\sigma_{\vec{l}}$ values

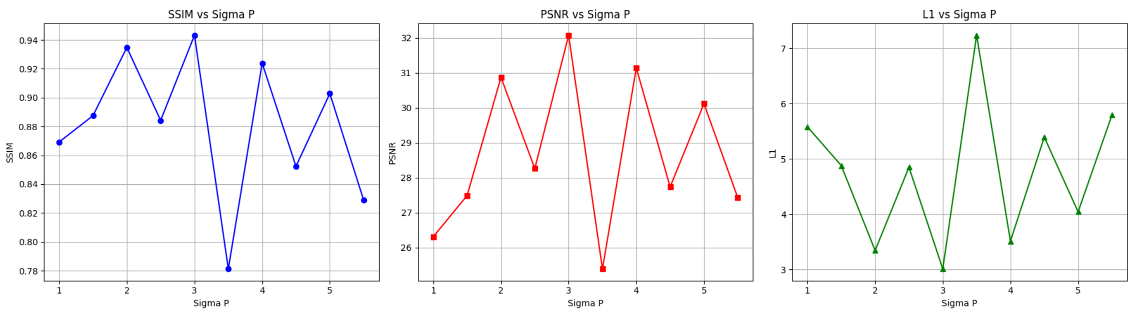


Figure 4.3: Graphs of metric values for different σ_p values

Figure 4.4: Images obtained while varying σ_l

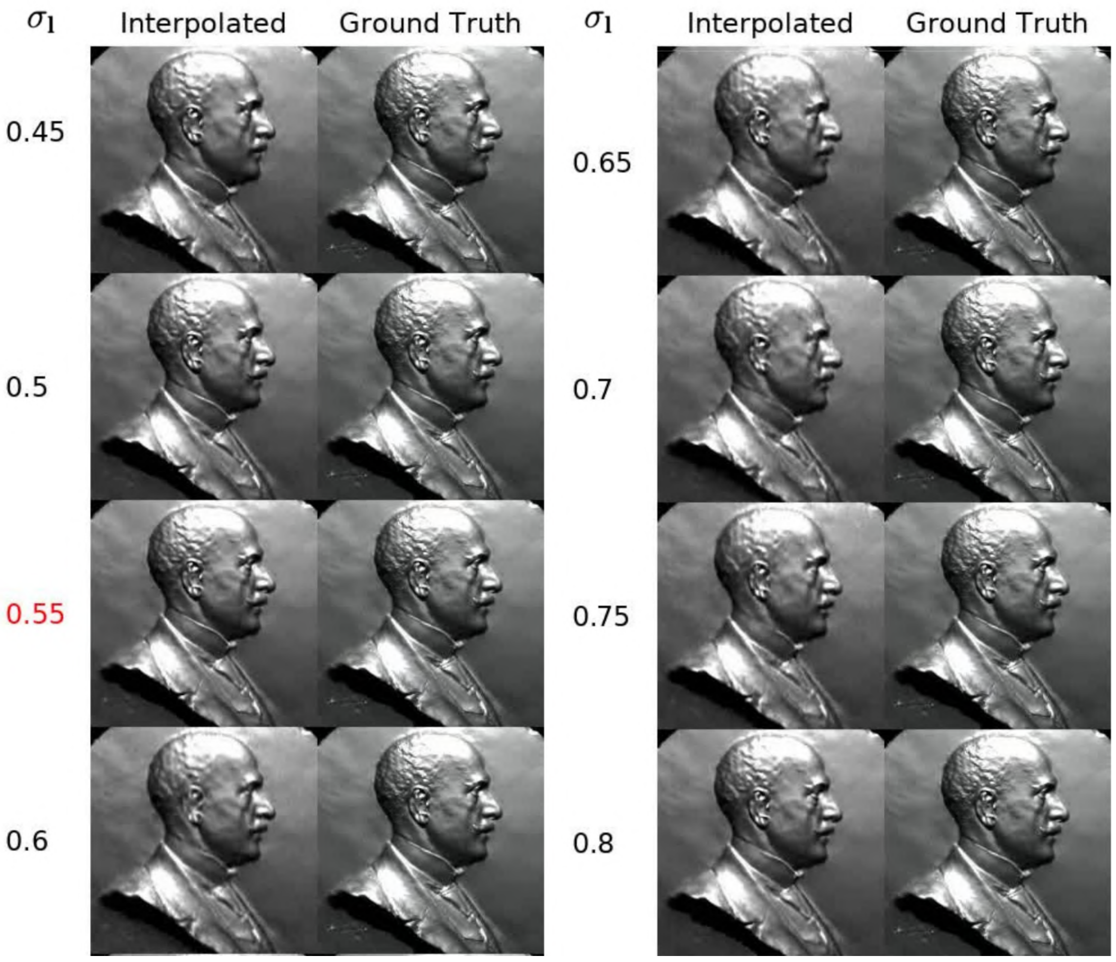
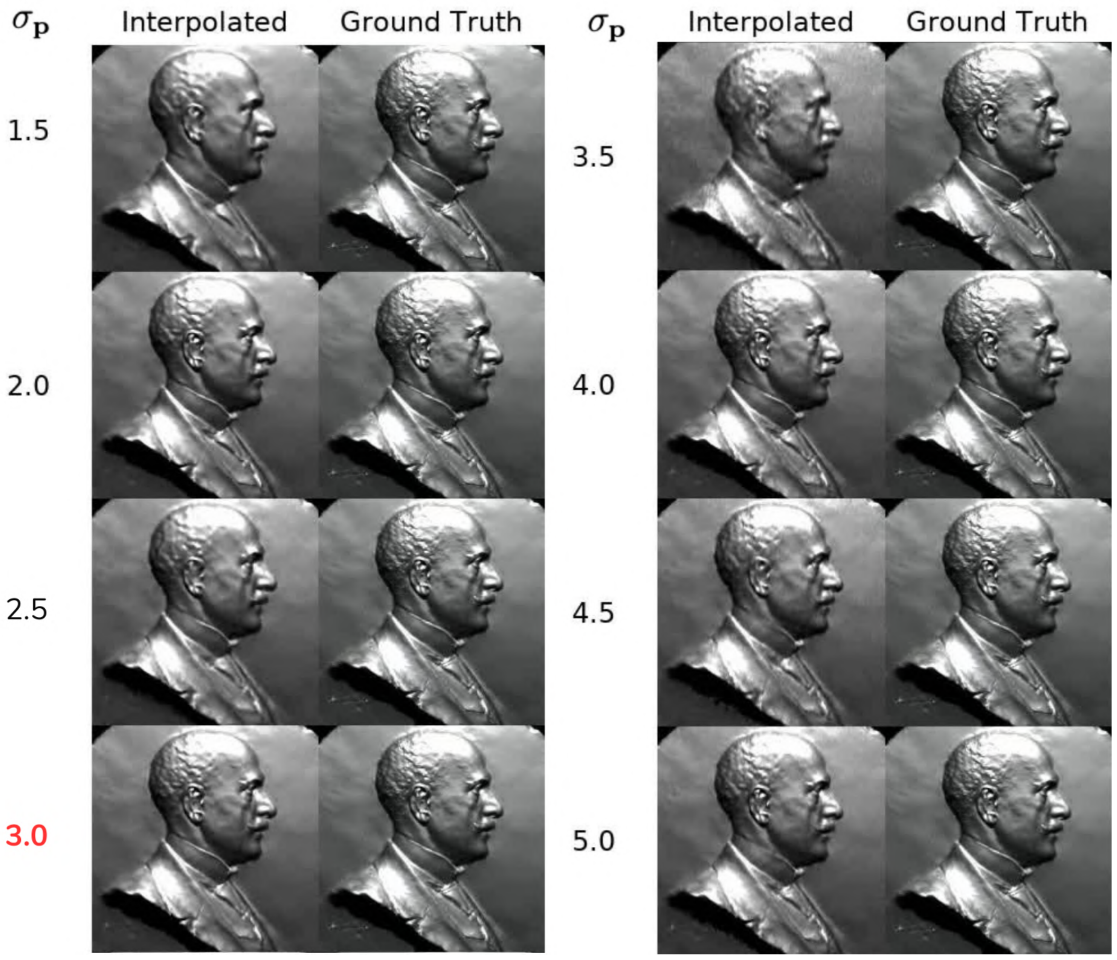


Figure 4.5: Images obtained while varying σ_p



4.3 Methods Comparison

Our model was compared with the PCA model described in [Chapter: 2.4.2] using a series of validation light directions, the outcome is presented in [Figure: 4.6] for our internal dataset and [Figure: 4.7] for the SynthRTI dataset. Although the images produced by both models are very similar to the ground truth, a side-by-side comparison in both comparisons reveals that our model's images are slightly blurred than those generated by the PCA model. However, it's worth noting that the shadows in our images more accurately reflect the ground truth than those in the PCA model's images, this is especially visible in the last three light directions of

[Figure: 4.6], where shadows are more visible. The final size of our model, $\approx$ 3MB, is comparable with the PCA-compressed data from the PCA model, $\approx$ 2.5MB plus the model dimensions. This indicates that our solution is also viable for practical applications.

Figure 4.6: Comparision between our neural model and the PCA model [1]

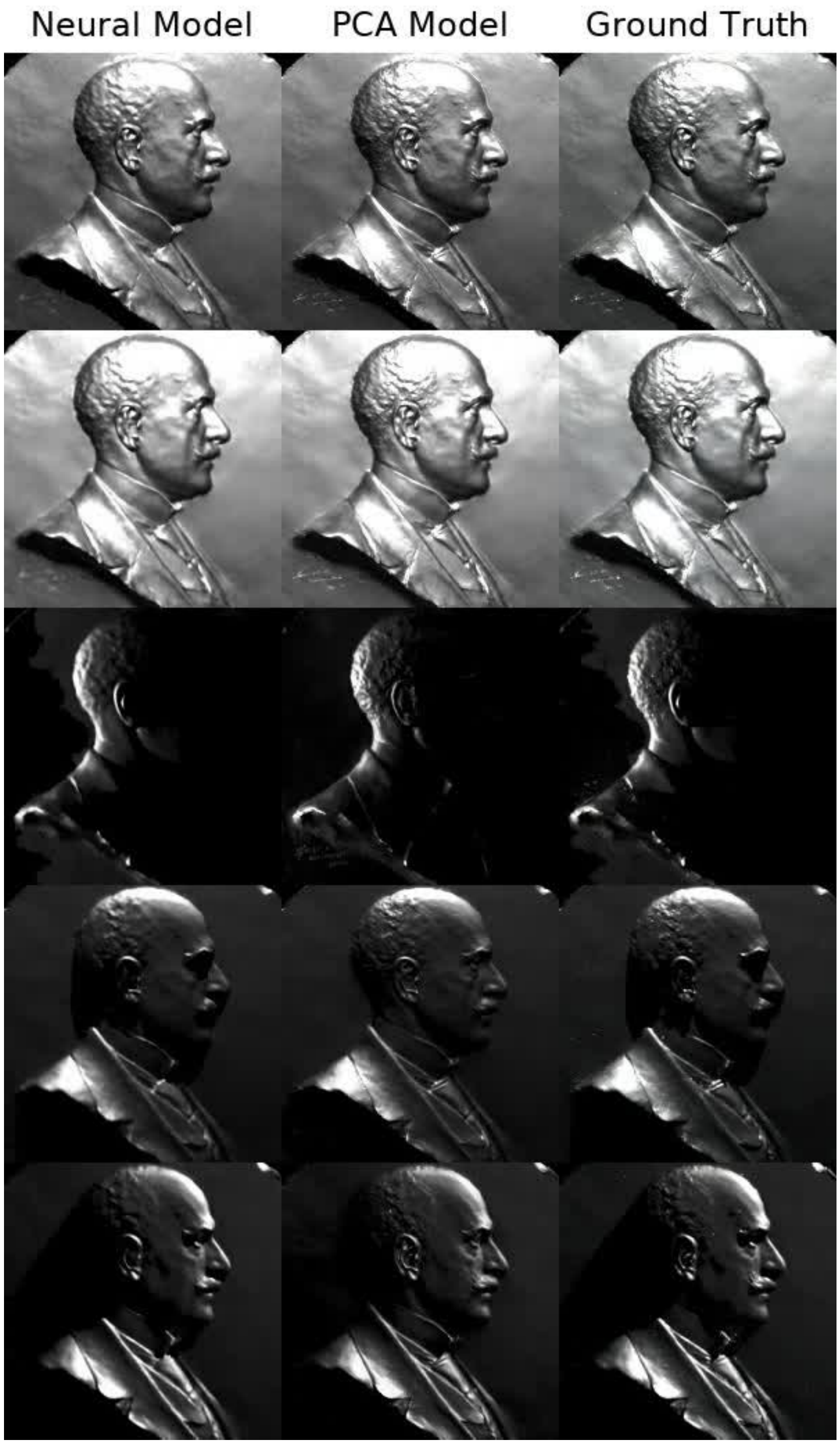
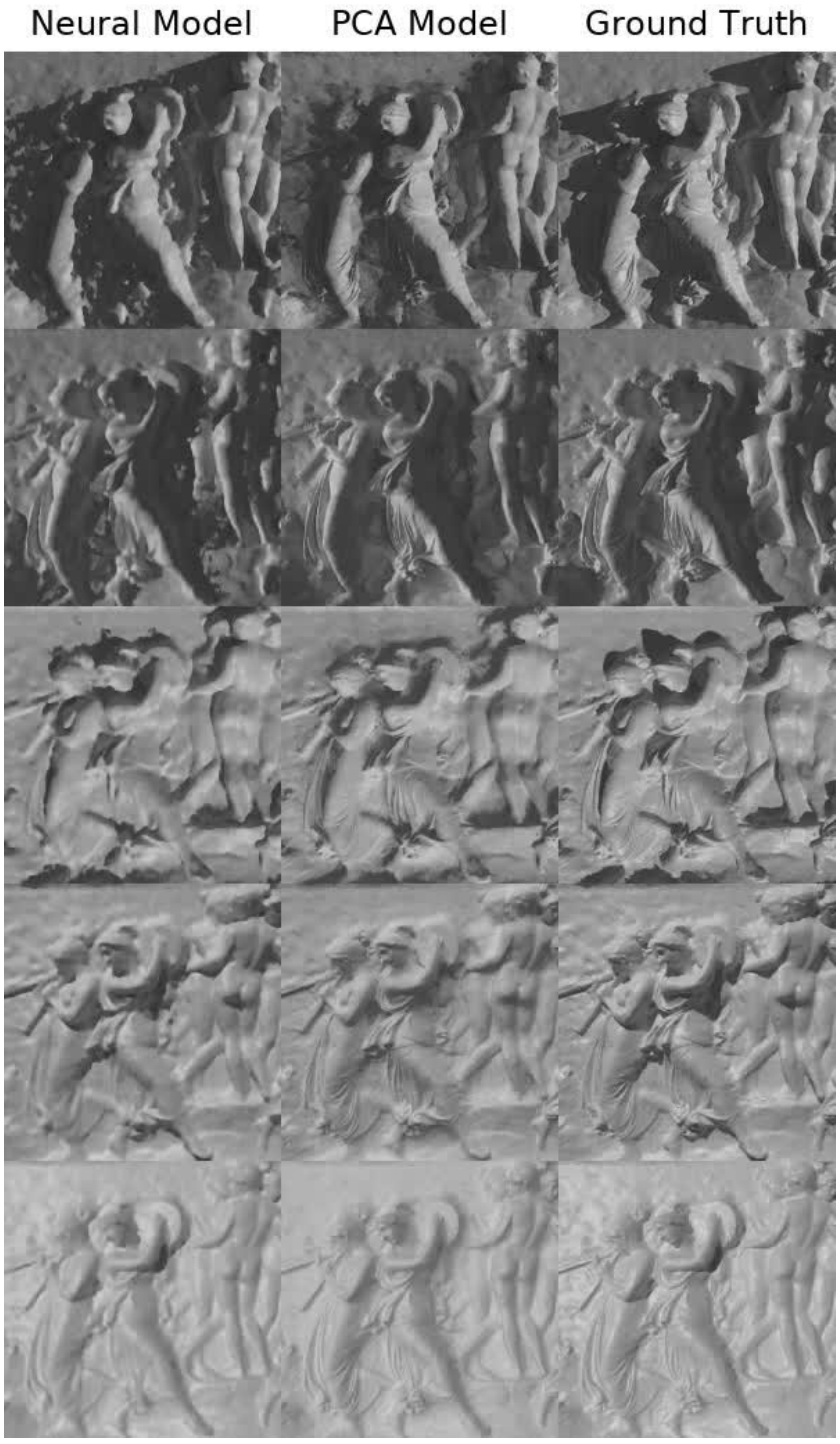


Figure 4.7: Comparision between our neural model and the PCA model [1] on the SynthRTI dataset



4.3.1 Training the model with fewer images

We tested how our model would perform if we removed an increasing number of images n_i from the dataset, the results are available in [Table: 4.3][Graphs: 4.8][Figure: 4.10], where n_i is the number of images in the dataset. [Table: 4.4][Graphs: 4.9] and [Figure: 4.11] show the same experiment on the SynthRTI [2] dataset (Single, Object 2, Material 3). This is a useful test to see how well the model performs with fewer images in the test set. As you can see from the images and the data, the model performance is stable and produces good results until $\approx 70\%$ of the images remain in our internal dataset and $\approx 50\%$ of the images remain in the SynthRTI dataset, afterwards the results starts to drop in quality drammatically. Consequently we can say that the model should perform well starting from roughly 60 images in our internal dataset and around 35 images for the SynthRTI dataset. Using more than that amount of images in the train set does not seem to improve the final output by a noticeable margin. It's important to note that the removal of the images from the dataset should done as fairly as possible, meaning that we should remove light direction randomly, and not sequentially, otherwise the model could be biased towards some light directions and hostile towards others.

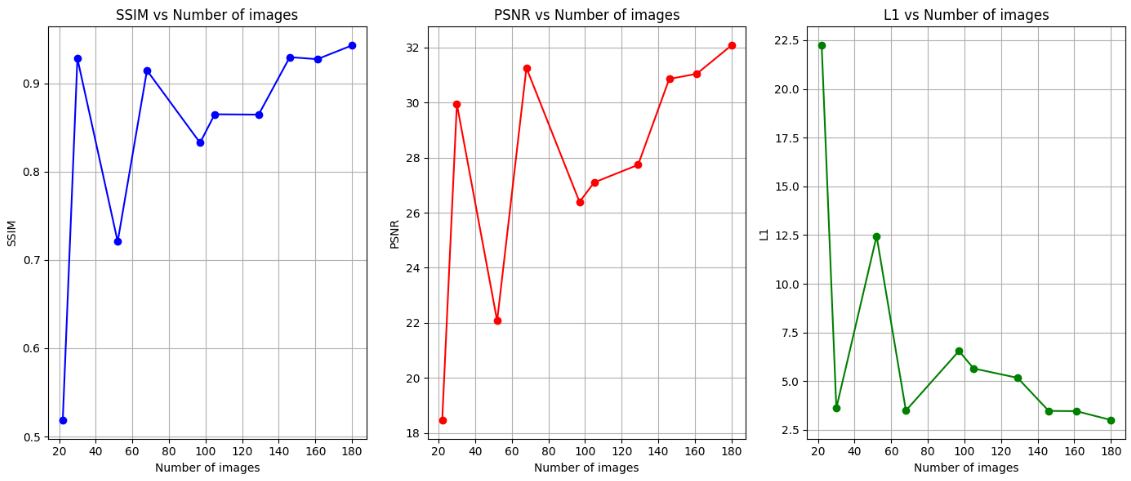


Figure 4.8: Graphs for the metric values for training the model on fewer images

$\mathbf{n_i}$	SSIM	PSNR	L1
180	0.9432	32.0739	3.0084
161	0.9276	31.0415	3.4642
146	0.9299	30.8555	3.4752
129	0.8648	27.7368	5.1725
105	0.8652	27.0980	5.6514
97	0.8328	26.3895	6.5498
68	0.9149	31.2575	3.4917
52	0.7209	22.0733	12.4447
30	0.9284	29.9431	3.6158
22	0.5187	18.4719	22.2322

Table 4.3: Metric values for training the model on fewer images

$\mathbf{n_i}$	SSIM	PSNR	L1
49	0.8225	27.2412	7.9401
47	0.8206	27.2523	8.0411
38	0.7980	26.1335	9.8490
32	0.7873	25.8050	9.9068
29	0.7244	22.1656	14.9095
24	0.7402	24.3924	13.9419

Table 4.4: Metric values for training the model on fewer images of SynthRTI

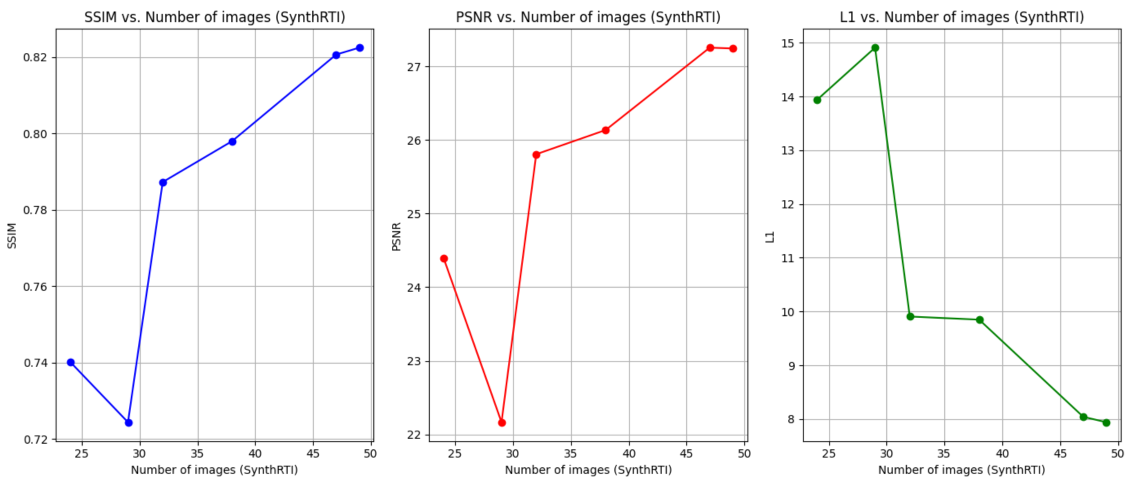


Figure 4.9: Graphs for the metric values for training the model on fewer images of SynthRTI

Figure 4.10: Images obtained while training the model on fewer images from the dataset

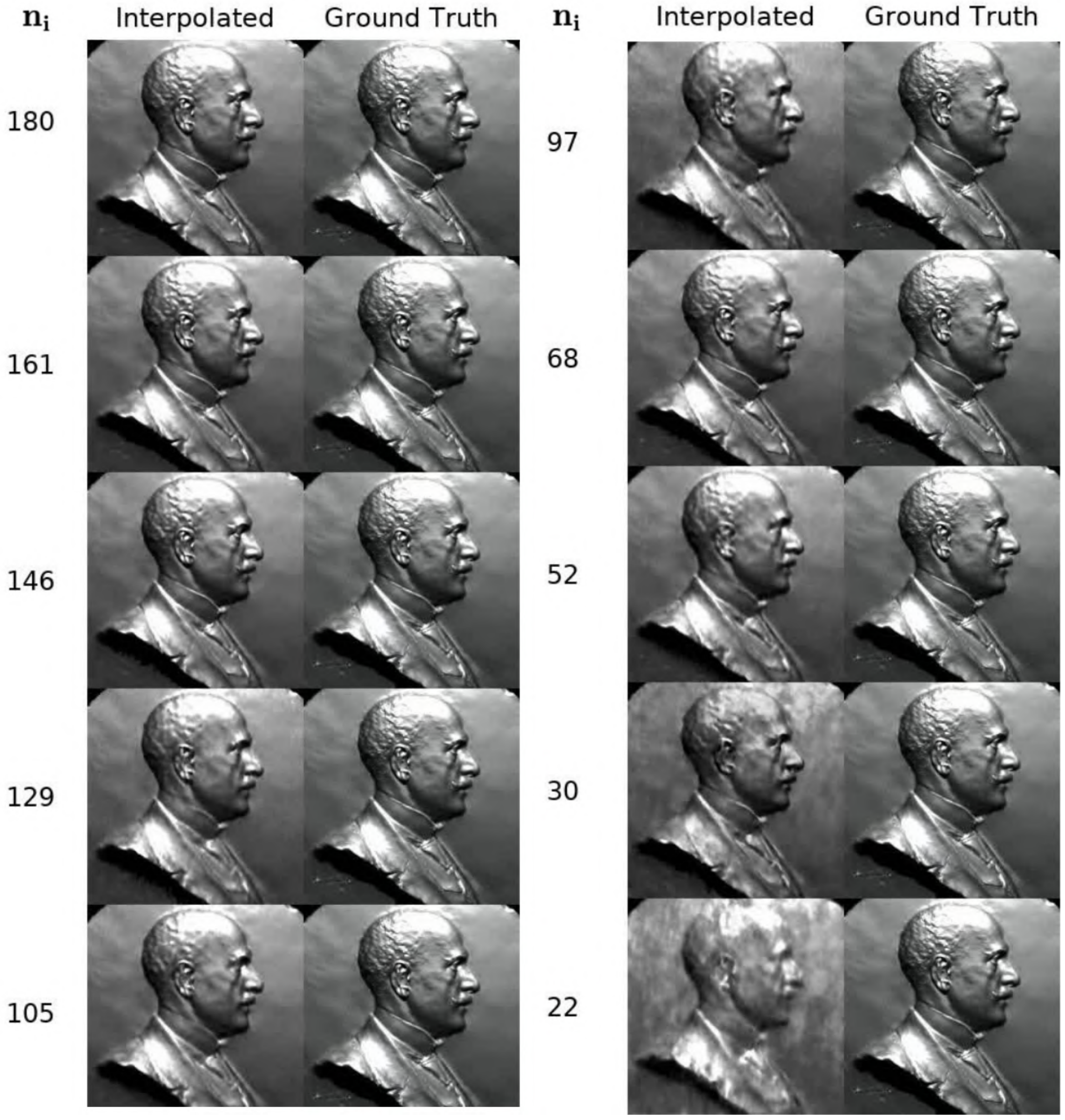
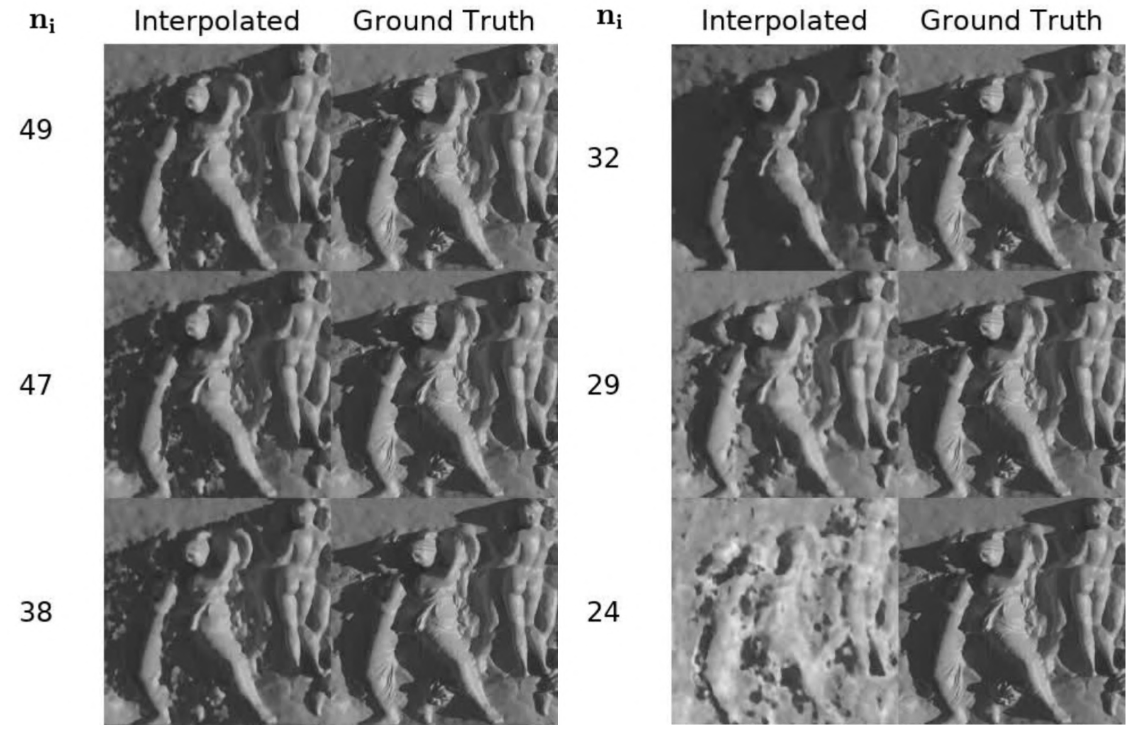


Figure 4.11: Images obtained while training the model on fewer images from the SynthRTI dataset



Chapter 5

Conclusions

In this thesis, we have proposed a new neural network with an Implicit Neural Representations to estimate the light transport function and interactively relight the scene in a photorealistic way [Chapter: 3.2.4] without requiring any additional data to relight the final scene. We have compared our results with existing state-of-the-art methods and demonstrated the feasibility of the approach [Chapter: 4.3]. Overall our results suggest that further investigation of INRs, in RTI applications, could prove successful. Our work could be expanded in many different ways, from using our model to do supersampling of the images by passing non-integers coordinates to overcome the camera limits, to using other functions like Discrete Cosine Transform (DCT) instead of the Fourier expansion on the model's input [Chapter: 3.2.4].

Bibliography

[1] **On-the-Go Reflectance Transformation Imaging with Ordinary Smartphones**
Pistellato, M., Bergamasco, F. (2023)
In: Karlinsky, L., Michaeli, T., Nishino, K. (eds) Computer Vision – ECCV 2022 Workshops. ECCV 2022. Lecture Notes in Computer Science, vol 13801. Springer, Cham.
https://doi.org/10.1007/978-3-031-25056-9_17

[2] **Neural reflectance transformation imaging**
Tinsae G. Dulecha, Filippo A. Fanni, Federico Ponchio, Fabio Pellacini & Andrea Giachetti
Vis Comput 36, 2161–2174 (2020),
https://link.springer.com/article/10.1007/s00371-020-01910-9

[3] **Low Cost Heritage Imaging Techniques Compared**
Caine, M. & Maggen, M.
(2017,7)

[4] **Digital Sensoriality: The Neolithic Figurines from Koutroulou Magoula**
Papadopoulos, C., Hamilakis, Y., Kyparissi, N. & Diaz-Guardamino, M.
Cambridge Archaeological Journal.
(2019,3)

[5] **Reflection Transformation Imaging and Virtual Representations of Coins from the Hospice of the Grand St. Bernard**
Mudge, M., Voutaz, J., Schroer, C. & Lum, M.
Proc. VAST. pp. 29-39 (2005,1)

[6] **New Reflection Transformation Imaging Methods for Rock Art and Multiple-Viewpoint Display**
Mudge, M., Malzbender, T., Schroer, C. & Lum, M.
Proceedings Of The7th International Symposium On Virtual Reality, Archaeology And Cultural Heritage (VAST2006). pp. 195-202 (2006,1)

[7] **Applications of Reflectance Transformation Imaging (RTI) to the study of bone surface modifications**
Newman, S.
Journal Of Archaeological Science. 53 pp. 536-549 (2015),
https://www.sciencedirect.com/science/article/pii/S0305440314004269

[8] **New applications of photogrammetry and reflectance transformation imaging to an Easter Island statue**
Miles, J., Pitts, M., Pagi, H. & Earl, G.
Antiquity. 88 pp. 596-605 (2014,6)

[9] **Underwater reflectance transformation imaging: A technology for in situ underwater cultural heritage object-level recording**
Selmo, D., Sturt, F., Miles, J., Basford, P., Malzbender, T., Martinez, K., Thompson, C., Earl, G. & Bevan, G. .
Journal Of Electronic Imaging. 26 pp. 011029 (2017,2)

[10] **Digital imaging and prehistoric imagery: A new analysis of the Folkton Drums**
Jones, A., Cochrane, A., Carter, C., Dawson, I., Diaz-Guardamino, M., Kotoula, E. & Minkin, L.
Antiquity. 89 pp. 1083-1095 (2015,10)

[11] **Combining RTI & SFM. A Multi-Faceted approach to Inscription Analysis**
Altaratz, D., Caine, M. & Maggen, M.
(2019,5)

[12] **A Benchmark Dataset and Evaluation for Non-Lambertian and Uncalibrated Photometric Stereo**
Shi, B., Mo, Z., Wu, Z., Duan, D., Yeung, S. & Tan, P.
IEEE Transactions On Pattern Analysis And Machine Intelligence. 41, 271-284 (2019)

[13] **Multiquadric equations of topography and other irregular surfaces**
Hardy, R.
Journal Of Geophysical Research. 76, 1905-1915 (1971)

[14] **Robust estimation of surface properties and interpolation of shadow/specularity components**
Drew, M., Hel-Or, Y., Malzbender, T. & Hajari, N.
Image And Vision Computing. 30, 317-331 (2012),
https://www.sciencedirect.com/science/article/pii/S0262885612000273

[15] **Introduction to radial basis function networks**
Orr, M. & Others
Technical Report, center for cognitive science, University of Edinburgh . . . ,1996

[16] **Using Radial Basis Function Networks for Function Approximation and Classification**
Yue Wu, Hui Wang, Biaobiao Zhang, K.-L. Du
International Scholarly Research Notices, vol. 2012, Article ID 324194, 34 pages, 2012.
https://doi.org/10.5402/2012/324194

[17] **Image-Based Empirical Information Acquisition, Scientific Reliability, and Long-Term Digital Preservation for the Natural Sciences and Cultural Heritage**
Mudge, M., Malzbender, T., Chalmers, A., Scopigno, R., Davis, J., Wang, O., Gunawardane, P., Ashley, M., Doerr, M., Proenca, A. & Barbosa, J.
Eurographics 2008 - Tutorials. (2008)

[18] **A Novel Hemispherical Basis for Accurate and Efficient Rendering**
Gautron, P., Krivanek, J., Pattanaik, S. & Bouatouch, K.
Eurographics Workshop On Rendering. (2004)

[19] **Discrete modal decomposition: a new approach for the reflectance modeling and rendering of real surfaces**
Pitard, G., Le Goïc, G., Mansouri, A., Favrelière, H., Desage, S., Samper, S. & Pillet, M.
Machine Vision And Applications. 28 pp. 607-621 (2017)

[20] **Discrete Modal Decomposition for surface appearance modelling and rendering**
Pitard, G., Le Goïc, G., Favreliere, H., Samper, S., Désage, S. & Pillet, M. (2015,6)

[21] **Matrix Computations**
Golub, G., van Loan, C.
Johns Hopkins University Press, Baltimore, 1989

[22] **Image based relighting using neural networks**
Ren, P., Dong, Y., Lin, S., Tong, X. & Guo, B.
ACM Trans. Graph.. 34 (2015,7),
https://doi.org/10.1145/2766899

[23] **Neural BTF Compression and Interpolation**
Rainer, G., Jakob, W., Ghosh, A. & Weyrich, T.
Computer Graphics Forum (Proceedings Of Eurographics). 38 (2019,3)

[24] **Implicit Neural Representations with Periodic Activation Functions**
Sitzmann, V., Martel, J., Bergman, A., Lindell, D. & Wetzstein, G.
Advances In Neural Information Processing Systems. 33 pp. 7462-7473 (2020), https://proceedings.neurips.cc/paper_files/paper/2020/file/53c04118df112c13a8c34b38343b9c10-Paper.pdf

[25] **Learning Deep Architectures for AI**
Bengio, Y.
Foundations And Trends® In Machine Learning. 2, 1-127 (2009)
http://dx.doi.org/10.1561/2200000006

[26] **Deep Convolutional AutoEncoder-based Lossy Image Compression**
Cheng, Z., Sun, H., Takeuchi, M. & Katto, J.
2018 Picture Coding Symposium (PCS). pp. 253-257 (2018)

[27] **Extracting and composing robust features with denoising autoencoders.**
Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.
Proceedings Of The 25th International Conference On Machine Learning. pp. 1096-1103 (2008)
https://doi.org/10.1145/1390156.1390294

[28] **Adam: A Method for Stochastic Optimization**
Kingma, D. & Ba, J.
(2017)

[29] **Perceptual Losses for Real-Time Style Transfer and Super-Resolution**
Johnson, J., Alahi, A. & Fei-Fei, L.
(2016)

[30] **Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains**
Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. & Ng, R.
(2020)
https://arxiv.org/abs/2006.10739

[31] **SynthRTI Github repository**
https://github.com/Univr-RTI/SynthRTI

[32] **Blender**
https://www.blender.org/

[33] **SketchFab**
https://sketchfab.com/

[34] **Photometric Method for Determining Surface Orientation from Multiple Images**
Woodham, R.J. 1980
Optical Engineerings 19, I, 139-144.

[35] **RTI Fixed Light Dome**
http://www.timzaman.nl/rti-dome

[36] **Polynomial texture maps**
Malzbender, T., Gelb, D. & Wolters, H.
Proceedings Of The ACM SIGGRAPH Conference On Computer Graphics. 2001 pp. 519-528 (2001,8)

[37] **Audacity**
https://www.audacityteam.org/

[38] **FFmpeg**
https://ffmpeg.org/

[39] **Tensorflow.js**
https://www.tensorflow.org/js

[40] **OpenCV**
https://opencv.org/

[41] **SciPy**
https://scipy.org/

[42] **NumPy**
https://numpy.org/

[43] **OpenCV - Create Calibration Pattern**
https://docs.opencv.org/4.x/da/d0d/ tutorial camera calibration pattern.html

[44] **PyTorch**
https://pytorch.org

[45] **Canny Edge Detector**
Canny, J. A computational approach to edge detection.
IEEE Transactions On Pattern Analysis And Machine Intelligence., 679-698 (1986)

[46] **Otsu Thresholding**
Otsu, N. A Threshold Selection Method from Gray-Level Histograms.
IEEE Transactions On Systems, Man, And Cybernetics. 9, 62-66 (1979)

[47] **Adaptive Thresholding Methods for Documents Image Binarization**
Bataineh, B., Abdullah, S., Omar, K. & Faidzul, M. *Pattern Recognition.* pp. 230-239 (2011)

[48] **An Interactive Method for Adaptive Acquisition in Reflectance Transformation Imaging for Cultural Heritage**
Khawaja, A., George, S., Marzani, F., Hardeberg, J. & Mansouri, A.
(2023,10)

[49] **Visualising an Egyptian Artefact in 3D: Comparing RTI with Laser Scanning**
Macdonald, L.
EVA'11 Proceedings Of The 2011 International Conference On Electronic Visualisation And The Arts, London, UK. (2011,1)

[50] **Polynomial texture mapping and 3D representations**
Macdonald, L. & Robson, S.
International Archives Of The Photogrammetry, Remote Sensing And Spatial Information Sciences - ISPRS Archives.
38 pp. 422-427 (2010,1)

[51] **Applications of reflectance transformation imaging for documentation and surface analysis in conservation**
Tamayo, S., Andrés, J. & Pons, J.
International Journal Of Conservation Science.
4 pp. 535-548 (2013,1)

[52] **Improved Positional Encoding for Implicit Neural Representation based Compact Data Representation**
Damodaran, B., Schnitzler, F., Lambert, A. & Hellier, P.
(2023)

[53] **LightBot: A Multi-Light Position Robotic Acquisition System for Adaptive Capturing of Cultural Heritage Surfaces**
Luxman, R., Castro, Y., Chatoux, H., Nurit, M., Siatou, A., Le Goïc, G., Brambilla, L., Degrigny, C., Marzani, F. & Mansouri, A.
Journal Of Imaging. 8 (2022)
https://www.mdpi.com/2313-433X/8/5/134

[54] **Objective evaluation of relighting models on translucent materials from multispectral RTI images**
Kitanovski, V. & Hardeberg, J.
Electronic Imaging. 33, 133-1-133-1 (2021),
https://library.imaging.org/ei/articles/33/5/art00004

[55] **CHI. Cultural heritage imaging website**
2019. [Online; accessed-March-2019]

[56] **RELIGHT: A compact and accurate RTI representation for the web**
Ponchio, F., Corsini, M. & Scopigno, R.
Graph. Models. 105 (2019,9),
https://doi.org/10.1016/j.gmod.2019.101040

Glossary

DMD Discrete Modal Decomposition. 19

FPS Frames Per Second. 31

GUI Graphical User Interface. 45

H-RTI Highlight Reflectance Transformation Imaging. 15

HSH Hemispherical Harmonics. 20, 21

INR Implicit Neural Representations. 4, 12, 13, 30, 60

MAE Mean Absolute Error. 41

MLIC Multi-Light Image Collection. 26–28

MLP Multilayer Perceptron. 28, 39

MSE Mean Square Error. 43

PCA Principal Components Analysis. 26, 28, 39

PSNR Peak Signal-to-Noise Ratio. 24, 25, 43

PTM Polynomial Texture Maps. 5, 38

RBF Radial Basis Function. 22, 38

ReLU Rectified Linear Unit. 39

RMS Root mean square error. 34

RTI Reflectance Transformation Imaging. 4–8, 14, 21–23, 38, 47, 48

SH Spherical Harmonics. 20

SSIM Structural Similarity index. 24, 25, 44

SVD Singular Value Decomposition. 38

Credits

I would like to thank all the people that helped me in this in my university journey like my supervisors Filippo and Mara; my colleagues that studied with me Giulio, Giovanni, Dario and Francesco and all others friends outside of university that cheered me on (sometimes by mocking me) and waited patiently until the end. I would also like to thank my family for supporting me and my choices throughout my whole education. A special thank goes to Pamela for being my life's lighthouse these past few years.