Technische Universität Kaiserslautern

Master Thesis

Improved Normal Inference from Calibrated Illuminated RGBD Images

Author:
Jingyuan Sha

Supervisors:
Prof. Dr. Didier Stricker
M. Sc. Torben Fetzer

Augmented Vision
Department of Computer Science

August 9, 2022

Declaration of Authorship

I, Jingyuan Sha, declare that this thesis titled, "Improved Normal Inference from Calibrated Illuminated RGBD Images" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Abstract

Augmented Vision
Department of Computer Science

Master of Science

Improved Normal Inference from Calibrated Illuminated RGBD Images

by Jingyuan Sha

In this work, a learning based approach for surface normal estimation based on illuminiated calibrated RGB-D images is proposed.

This approach extends the geometry based approach by techniques of photometric stereo, which uses calibrated illuminated images for normal estimation. The two types of data gives a superior performance on normal inference task. One of the key points of our approach is, the proposed method require only one light source for each scene capture instead of multiple light sources, which can be very easy to configure in the practice.

Our model is trained on a gated convolution neural network (Trip-Net). The gated convolution design is intended to handle the missing pixels in the depth map, which is a common defect of the depth sensor. We also created a dataset of 55 point clouds called *Synthetic-50-5* for training. We trained our model on this synthetic dataset and then applied it to a real dataset. Finally, a qualitative and quantitative evaluation is performed.

Contents

Chapter 1

Introduction

In 3D computer vision tasks, the *point cloud* (i.e., an unstructured set of 3D points representing the surfaces in the scene) is a common input file to describe the object surface. Based on the geometry information in the point cloud, we can calculate the corresponding surface normal, which is an important feature in computer vision. The surface normal can be used for surface reconstruction, shading generation and other visual effects. The common method for determining the surface normal from the point cloud is the optimization approach, which considers the neighboring points in the same plane and calculates the surface normal.

However, for single input data with only one shot, such as the data acquired by a depth camera, the point cloud is converted from a depth map and the given calibration information. In the case of active depth sensors, the depth map is usually only semi-dense due to optical noise and reflections in dark and shiny areas of the object surface, missing many pixels and holes. This limits the neighborhood-based approach because the missing data results in a smaller number of neighbors for each point. However, recent advances in the field of image inpainting have motivated researchers to use deep neural networks to recover the missing regions in the input data and achieve good performance. For example, [35] uses a gated convolution network to recover user-defined removed regions in the images. [16] uses a normalized convolution network to extend the semi-dense depth map to a fully dense depth map. In both cases, incomplete input data is used and converted to a fully dense output. Therefore, a Deep Learning based approach can be a way to deal with semi-dense depth maps for surface normal estimation tasks.

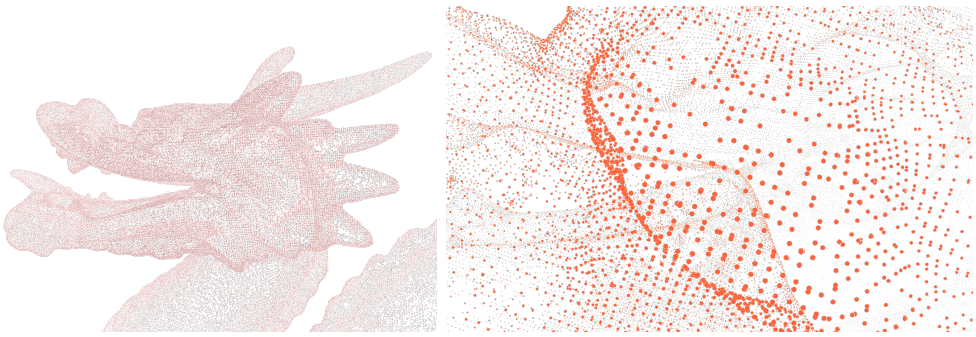


Figure 1.1: Left: A portion of the point cloud of the *dragon* object. Right: Zoom in of the point cloud.

Photometric stereo is another approach in computer vision that can be used for estimating surface normals, where the normals are derived from appearance variations under different illuminations. The corresponding methods are usually solved using the least squares problem, which is based on the point-wise image formation

model [11] and on the assumption of the Lambert surface. However, the Lambert surface generally cannot account for global illumination effects, which is an unavoidable problem due to light interactions.

Both of the above directions have been widely explored using deep learning approaches. However, the approach that considers both geometry and illuminated information for normal estimation remains insufficient. We present a Deep Learning based approach that considers both geometric and photometric information for normal inference. For practical reasons, we consider a semi-dense depth map with only one additional illuminated image based on the calibrated camera as input data to find a compromise between the two methods.

One of our challenges is to apply the semi-dense input data to the neural network in an image-aware manner. *U-Net* is a good network for computer vision tasks with up-sampling requirements. We modified it with a gated activation unit called *gated convolution layer (gconv)* to handle semi-dense input data. Another challenge is to merge the different feature maps of the input data to cooperatively improve the surface normal inference task. We propose a *multi-fusion* scheme for integrating separate feature maps. We will show that this *multi-fusion* design can be leveraged by combining different input data types and improving the performance of surface normal inference. To train our network, we create a synthetic RGB-D image dataset by using a light source and ambient light illumination to simulate application scenarios. To simulate depth maps in the real world, we added uniformly distributed drop-out noise to the input data, with the noise density controlled by a parameter.

We trained our models on our proposed synthetic dataset *synthetic-50-5* and evaluated them on 6 different metrics. The results show that our approach based on calibrated illuminated RGB-D images further improves the normal accuracy. We also applied our model to a real dataset and compared it with the optimization-based approach.

The structure of the paper is as follows: In Chapter 2, we briefly discuss related work on the normal inference task. Chapter 3 briefly introduces a traditional optimization-based approach and photometric stereo, and then proposes two learning-based approaches for surface normal estimation. Chapter 4 presents the dataset created for the training work and its preprocessing. Chapter 5 analyzes the experiments and provides both qualitative and quantitative evaluations of the models. Chapter 6 is the summary of the entire thesis.

Chapter 2

Related Work

Surface normal inference is a widely researched topic and can be solved in several ways. Traditionally, it can be derived from a point cloud. There are many optimization-based approaches to computing surface normals pixel-by-pixel using neighborhood information. [15] gives a comparison between some of these approaches. [10] proposed a method of using local neighbors with an interest parameter p and derives the surface normal from the eigenvectors of the corresponding covariance matrix. These approaches work well for dense input data based on a well-chosen window size. However, in many practical datasets such as NYU Depth Dataset V2 [28] and KITTI-Depth[32], the point clouds are often converted from depth maps. But the depth maps are usually very noisy and contains many missing pixels and holes in dark, shiny, or transparent areas. This poses more of a challenge for neighborhood-based approaches, as neighborhood information is not available in many of these areas.

To solve this problem, some approaches focus on improving the depth map by estimating missing regions. [34] proposed an approach that considers the depth distributions of neighboring regions to fill the holes in the depth map. [4] uses a deep learning-based approach for painting semi-dense depth maps, which mainly uses the normalized convolution proposed by [16]. This approach considers a 0-1 mask to distinguish valid and invalid data points. [6] improved this approach by proposing an input confidence estimation network to replace the binary mask with an estimated confidence mask. [12] also uses a normalized convolution for inpainting the depth map and also uses an RGB image as a guidance.

On the other hand, some of the approaches use RGB-D images to estimate the surface normal or use RGB images to estimate the fully dense depth map. [3] proposed a two-level network for predicting the depth map based on RGB images, where the global and local features of the object are considered separately. [17] also predicts the depth map from the image, using a residual network for feature extraction and developing an up-sampling part with the un-polling layers. [18] proposed a method that adaptively generates depth map prediction information based on RGB images. [7] uses RGB-D images to estimate surface normals. It proposes a method for learning discriminative and geometrically informative primitives from RGB-D images, which is then used to recover the surface normals of a scene from a single image. [25] uses ResNet [8] to derive a coarse surface normal from an RGB image, and refines it using a depth map based on the [7] method. [36] derives surface normals based on an RGB-D image using the *U-Net* architecture.

However, these approaches mainly rely on RGB images as input for estimating the normal or the full depth map. The applied scenarios are mainly indoor scenes such as bedrooms or offices, where the focus is not on the fine details of the object surface, but only on the main directions of each area, such as the directions of the floor, walls, or seats, etc as shown in Figure 2.1

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

Figure 2.1: (a)-(b): Output from the RGB camera (a) and depth camera (b) of a scene in *NYU Depth Dataset V2* [28]. (c)-(d): Output from the gray-scale camera (c) and depth camera (d) of a scene in our proposed dataset *synthetic-50-5*.

Photometric stereo is another approach that derives the surface normal from the BRDF-based surface model. The surface normal is related to the observed intensity, the direction of incident light, the direction of outgoing light, and the light intensity. Several light conditions are required to obtain an optimal solution. [13] uses a CNN-based approach to learn the relationship between photometric stereo input and the surface normal under hundreds of lighting conditions, which is also robust to the effects of global illuminations. [37] is based on an illumination interpolation network to use only a sparse set of lights for photometric stereo tasks.

Our approach uses an RGB-D image but known light direction and calibrated information for surface normal inference, which considers multiple input data types into account for the estimation.

Chapter 3

Approaches

The normal inference task estimates the surface normal of a 3D object, which is assumed that the object's surface is mathematically unknown. It means we cannot simply use the cross-product of two non-parallel surface vectors to find the normals. Instead, the surface is acquired from digital sensors under specific data types, which can be divided into two groups. The first is geometry data, which uses a point cloud to record the surface geometry information in 3D space. The approaches use point clouds for normal estimation is called geometry based approach. Another data type is the images, which record the surface reflectance and shading under a light condition. The surface normal is calculated based on specific light models like BRDF. The approaches use the relationship of reflectance and the shading to calculate the surface normal is called photometric stereo.

In this chapter, geometry and photometric stereo based approach are introduced separately, then we introduce a method to estimate the surface normal that utilize both kind of data types based on deep learning approaches.

3.1 Notations

The following standard notations are adopted in the following chapters of this thesis. The boldface capital letters stand for matrices, whereas the boldface lowercase letters stand for vectors. The Greek capital letters stand for planes. We use the subscript $i = \{1, 2, ...\}$ to denote the index of the observations, whereas the superscript to denote the dimension of the matrices or vectors, e.g. $\mathbf{n}^{3\times1}$ denotes a vector with dimension 3×1. We use W stands for width, H stands for height, and C stands for the number of channels.

3.2 Geometry based Normal Estimation

The geometry based normal estimation uses the point cloud of the object surface as input. The task can be stated as follows.

Given a structured point cloud $\mathbf{V}^{W\times H\times3}$, each point in the cloud corresponds to a position on an object surface. We want to calculate the corresponding normal map $\mathbf{N}^{W\times H\times3}$. The normal map is simply a matrix representation of the normals of all the points in the point cloud. A normal $\mathbf{n}^{3\times1}$ at point $\mathbf{p}^{3\times1} \in \mathbf{V}$ is a unit vector with its direction point outward of the surface and perpendicular to the tangent plane of the surface at point $\mathbf{p}$.

The idea behind the neighbor based method is to fit a plane $\tilde{\Pi}$, which uses the k neighbors $\mathbf{p}_1, ..., \mathbf{p}_k \in \mathbb{R}^3$ of the point $\mathbf{p}$, then calculate the normal $\tilde{\mathbf{n}}$ of the plane $\tilde{\Pi}$. The $\tilde{\mathbf{n}}$ is considered as an approximation of the normal $\mathbf{n}$ of the tangent plane Π.

It is under the assumption that the point and its neighbors are located in the same plane Π, which is exactly the tangent plane of the surface at point $\mathbf{p}$. This is usually not held for most of the surface since the surface is usually not flat but with a degree of curvature. But if k in a suitable scale and the point cloud is dense enough, we can approximately consider all the k neighbor points in a small range on the same tangent plane. Then we have an approximation $\mathbf{n} \approx \tilde{\mathbf{n}}$, where $\tilde{\mathbf{n}}$ denotes the estimated normal. Similarly, the estimated tangent plane $\tilde{\Pi} \approx \Pi$.

Based on the assumption mentioned above, the normal inference task is trivial since then the surface is mathematically describable. We can generate the approximate tangent plane at any point with its neighbors, then the normal $\tilde{\mathbf{n}}$ of the tangent plane can be derived from the following equations

$$\tilde{\mathbf{n}} \cdot \mathbf{v}_{ij} = 0$$

where

$$\mathbf{v}_{ij} = \mathbf{p}_i - \mathbf{p}_j, \text{for } i, j \in \{1, 2, ..., k\}, i \neq j$$

To find a normal, we need at least 3 neighbors which are not in the same line to solve the equation above. Calculate the normal for each point on the point cloud, then we can get the corresponding normal maps.

If we consider more than 3 neighbors, the points are usually not located on the same plane. In this case, the equation system is over-determined. In this case, we can use the optimization approach to find a plane that has the minimum distance to all the points in the 3D space. Let $\mathbf{A} \in \mathbb{R}^{(k)\times 3}$ denotes the vector matrix vertically stacked by the k vectors we found on the approximated tangent surface.

Then, the equation system is converted to a minimum problem

$$\begin{aligned} \min \quad & \|A\mathbf{n}\|_2^2 \\ \text{s.t.} \quad & \|\mathbf{n}\|_2^2 = 1 \end{aligned} \tag{3.1}$$

where the extra constraint is added to avoid trivial solutions and let the normal be a unit vector.

The problem mentioned above can be solved by singular value decomposition(SVD), where

$$A = U\Sigma V^T$$

The last column of V is the equation system solution and consequently the surface normal.

The optimization-based approach using SVD is more robust compared to only choosing 3 points for normal calculation since the plane generated by 3 neighbors does not guarantee to be a good approximation to the actually tangent plane. However, the neighbor size k is a key parameter of this approach. For the smooth surface, a relatively bigger k can help the equation system to fit a plane that is closer to the tangent plane. For the sharp surface area, the k has to be small enough to ensure that the points it contains are still approximately in the same plane, otherwise the more included points are only the outliers for the equation solving.

At last, after solving the equation mentioned above, we should convert all the normals to the viewpoint $\mathbf{s}$ for a unitary reason. Thus the direction of a normal should be inverted if

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{s}) > 0 \tag{3.2}$$

Since the approach mentioned above is basically solved by SVD, thus for simplicity, this method will be named SVD in the rest of the thesis.

3.3 Photometric Stereo

The photometric stereo was initially introduced by [33], which is a completely different approach compared to geometry information based approach like SVD. It estimates the surface normal of the object by observing the object in the same position under different illuminated scenes. It is based on the fact that the light reflected by a surface is dependent on the surface normal and the light direction.

Before we introduce this approach, let us discuss what we can actually get from the images. Given an image $\mathbf{I}$, it can be decomposed into two parts, the reflectance $\mathbf{R}$ and the shading $\mathbf{S}$,

$$\mathbf{I} = \mathbf{R} \oplus \mathbf{S}$$

where $\oplus$ denotes the element-wise product. This decomposition of the image is based on the intrinsic image model, which was proposed by [1]. It interprets the observed image into a reflectance image and a shading image. Figure 3.1 gives a visualization of the decomposition.

Figure 3.1: Intrinsic image analysis of the Washington object. From left to right, original image, reflectance image, shading image, light image, normal image.

The equation can be further decomposed based on different surface models. If assume the object surfaces are Lambertian surfaces, i.e. the surface which reflects light in all directions, as shown in figure 3.2, the shading image can be decomposed as the product of the radiance of incoming light L_0, the cosine of the angle of incidence, which is the dot product of the surface normal $\mathbf{N}$ and the light source direction $\mathbf{L}$.

$$\mathbf{I} = \rho \odot (L_0 \mathbf{L} \cdot \mathbf{N})$$

note that each surface normal in the matrix $\mathbf{N}$ and light direction in matrix $\mathbf{L}$ is a unit vector thus they have only two degrees of freedom.

The equation can be further rearranged as follows

$$\mathbf{I} = (L_0 \rho \odot \mathbf{N}) \cdot (\mathbf{L})$$

Let $\mathbf{G} = L_0 \rho \odot \mathbf{N}$, the equation is simplified to

$$\mathbf{I} = \mathbf{G} \cdot \mathbf{L}$$

If we use a set of k images for the same scene that have been captured based on different light projections. Then, for each pixel (x, y) in a scene, an equation system

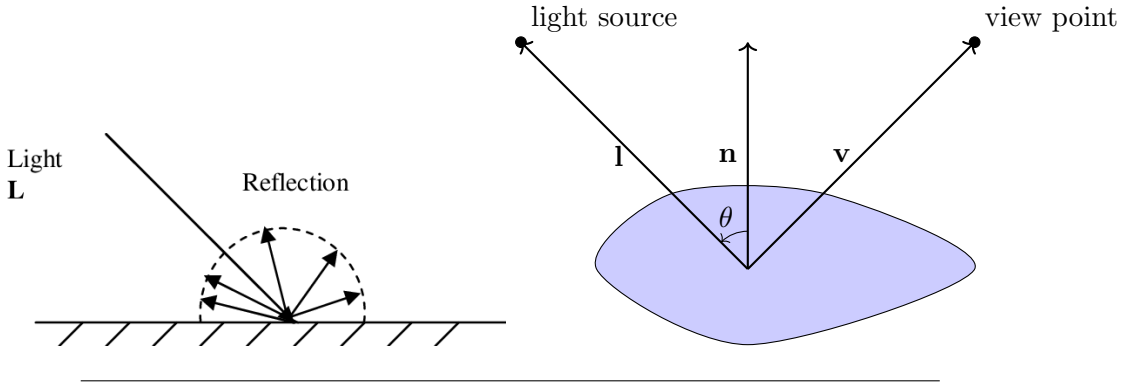


Figure 3.2: Left: the light reflection on a Lambertian surface. (image source [20]). Right: The surface normal, source light direction, and the viewpoint direction, where θ denotes the angle between light direction and the normal.

can be set up

$$\begin{pmatrix} \mathbf{l}_1^T \\ \mathbf{l}_2^T \\ \cdots \\ \mathbf{l}_k^T \end{pmatrix} \mathbf{G}(x,y) = \begin{pmatrix} \mathbf{I}_1(x,y) \\ \mathbf{I}_2(x,y) \\ \cdots \\ \mathbf{I}_k(x,y) \end{pmatrix}$$

For simplicity, $\mathbf{l}_i^T$ for $1 \leq i \leq k$ denotes the light direction at position (x, y) in the image k . The equation can be solved based on the least square methods.

First, we can get the albedo with the light intensity based on the fact that normal is a unit vector, thus we have

$$\|\mathbf{G}(x,y)\|_2 = \|L_0\rho(x,y)\mathbf{N}(x,y)\|_2 = L_0\rho(x,y)$$

Then the normal can be obtained as follows

$$\mathbf{N}(x,y) = \frac{\mathbf{G}(x,y)}{L_0\rho(x,y)}$$

We calculate the surface normal for each point to get the surface normal map.

This approach is also called shape from shading(SFS) [11]. It gets both normal maps and the corresponding albedo from a set of images under different light conditions. Since the light direction is used in the computation, the cameras and the light source position have to be calibrated beforehand.

The shape from shading approach takes at least three light positions into account for the normal estimation on one pixel. It is also common to use more than three light sources to cover most of the surface points as much as possible.

3.4 Gated Convolution Neural Network for Surface Normal Estimation

Recently, deep learning based method achieved great success for image processing. ([26], [30]). These network architectures use a batch of RGB / Gray-scale images as input and are employed for classification problems. Usually, the images are convoluted with a convolution layer and down-sampled with pooling layers. The outputs of the network consist of a single value to represent the index of the corresponding class ([30]) or with a set of values to represent the position of bounding boxes.([26]). However, in many other vision tasks like normal map inference, the output is demanded as the same shape as the input, whereas each pixel in the output image requires a prediction. Therefore the output is not only one or several labels but a similar size matrix as the input. In this case, the traditional network architecture using fully connections in the last layers for label prediction is not suitable anymore.

Recently, [27] proposed an architecture called UNet for biomedical image segmentations. The architecture is shown in Figure 3.3. This network has a very regular architecture for the data processing whereas down/up sampling part has a similar design. For the feature extraction part, as known as down-sampling, the architecture follows the traditional CNN architecture. For the up-sampling part, the network uses an architecture that is similar to down-sampling operations, which uses same times convolution operation but replaces the max pool layers with up-conv layers. An important design is the skip connection in the up-sampling part. The feature maps extracted in the down-sampling part will be concatenated with the feature maps in the up-sampling part. This helps the network to find the locations in the original image and use it to predict a sharp output.

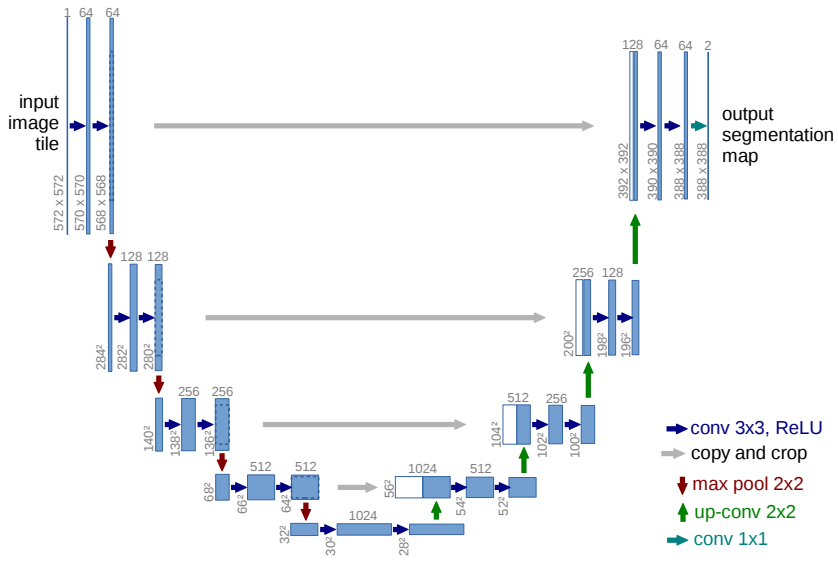


Figure 3.3: The structure of UNet. [27]

The UNet is based on standard convolution layers to construct the network. This is reasonable for image processing tasks with full-dense input since no missing pixels exist. However, some other input data like depth maps captured from the light scanners, they are not always fully dense data but equipped with lots of missing pixels and holes. In this case, we can still apply the standard convolution layers to extract the feature maps from the semi-dense depth map, but then it will be confused

by the valid pixels and the invalid pixels (the missing pixels) during the training, and consequently leads to blurriness and color discrepancy, as mentioned by [19]. Thus it is reasonable to find a way to distinguish the valid and invalid pixels during the training.

[5] use a binary mask to indicate valid pixels and further use normalized convolution as a substitution for the standard convolution layer in the training network. The normalized convolution is shown as follows

$$O(x,y) = \begin{cases} \dfrac{\Sigma_i^k \Sigma_j^k W(i,j) \odot I(x-i,y-j) \odot M(x-i,y-j)}{\Sigma_i^k \Sigma_j^k W(i,j) \odot M(x-i,y-j)}, & \text{if } \sum_i^k \sum_j^k M(i,j) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where k is the kernel size, (x, y) is the position in input, (i, j) is the displacement in kennel, M is the corresponding mask. A binary mask uses 1 to indicate valid pixels and 0 otherwise. $\odot$ denotes element-wise multiplication.

The normalized convolution layer added weight to the mask. However, initialization for the mask is still required, and the propagation of the mask remains a tricky task.

3.4.1 Gated Convolution

[23] proposed a gated activation unit to model more complex interactions compared to standard CNN layers, which was mainly inspired by the multiplicative units that exist in Long Short-Term Memory proposed by [9] and Rated Recurrent Unit (GRU) proposed by [2]. [35] employed the same gated unit and use it as a gated convolution layer for training tasks with in-complete input data, such as image inpainting tasks. In this gated convolution layer design, the mask used to distinguish the valid and invalid pixels are not predetermined but learned during the training.

The structure is shown in Figure 3.4. Instead of using a mask as input to indicate valid pixels, it employs a standard convolution layer to learn this mask directly from data, and use a sigmoid function as the activation function to indicate the confidence of the pixel validation. Meanwhile, another standard convolution layer placed aside to learn the feature maps with a ReLU /LeakyReLU activation functions. Therefore the mask and the feature of the input are both learned during the training. Then it imply element-wise multiplication with the feature map and the mask as the final feature map of this gated convolution layer.

Formally, the gated convolution is described as follows, the layer with input size (N, C_{in}, H, W) and output size $(N, C_{out}, H_{out}, W_{out})$:

$$o(N_i, C_{o_j}) = \sigma(\sum_{k=0}^{C_{in}-1} w_g(C_{o_j}, k) \star i(N_i, k) + b_g(C_{o_j})) * \phi(\sum_{k=0}^{C_{in}-1} w_f(C_{o_j}, k) \star i(N_i, k) + b_f(C_{o_j})) \tag{3.4}$$

where ϕ is LeakyReLU function, σ is sigmoid function, thus the output values are in range $[0, 1]$, $\star$ is the valid 2D cross-correlation operator, N is batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels, $w(C_{o_j}, k)$ denotes the weight of j-th output channel corresponding k-th input channel, $i(N_i, k)$denotes the input of i-th batch corresponding k-th input channel, $b(C_{o_j})$ denotes the bias of j-th output channel.

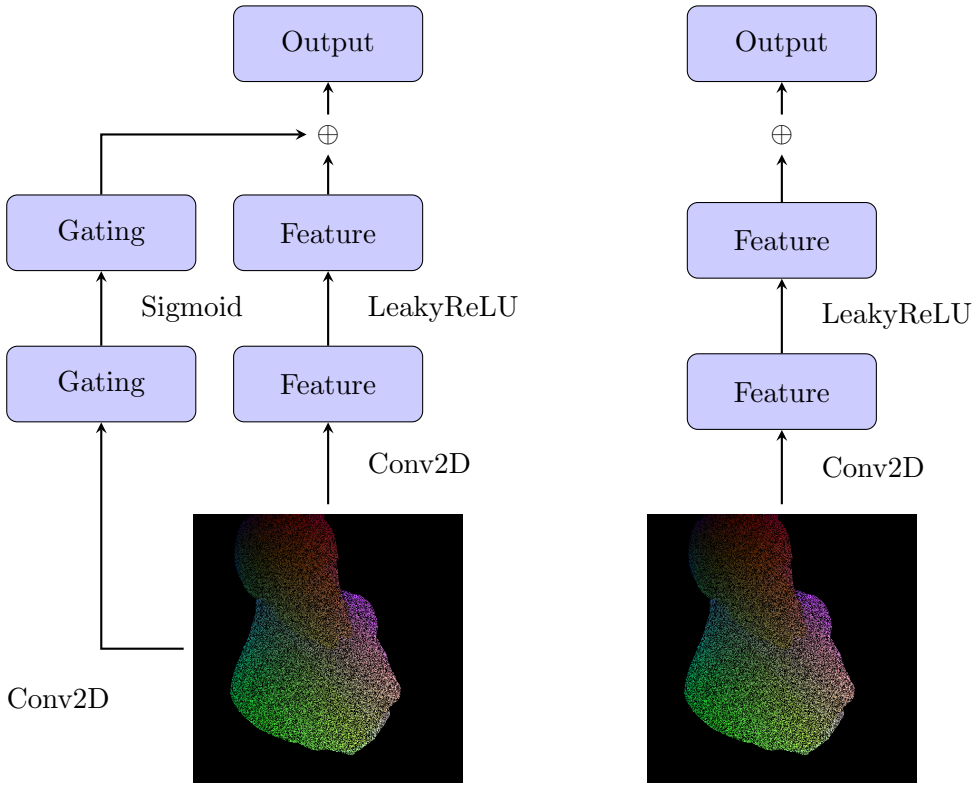


Figure 3.4: Left: Gated Convolution Layer, where $\oplus$ denotes element-wise multiplication. Right: standard convolution layer.

Thus instead of use a hard-coded mask, the network is able to learn the mask during the training.

3.4.2 GCNN Architecture

Based on the implementation mentioned above, we proposed a network that based on UNet proposed by [27] and replace the standard convolution layers with gated convolution layers using for semi-dense input normal inference task, called gated convolution neural network (GCNN), as shown in Figure 3.5.

In order to describe the network in a common way, the parameters of the network are represented by letters. The network is constructed by a downsampling part and an upsampling part. In the downsampling part, the input has shape $\mathbf{X}_{in} \in \mathbb{R}^{H \times W \times C_{in}}$ whereas output has shape $\mathbf{X}_{out} \in \mathbb{R}^{H \times W \times C_{out}}$ The input matrix goes through 3 times down-samplings. A down-sampling block has two gated convolution layers with stride (1,1) and an extra gated convolution layers with stride (2,2) to reduce the feature map resolution. Thus there are in total 3 gated convolution layers in each down-sampling operation. The total three times downsamplings extract the geometry features $\mathbf{X}_f$ from input matrix $\mathbf{X}_{in}$ (represented as a regression function f)

$$f : \mathbf{X}_{in} \rightarrow X_f$$

After the feature extraction, the network upsampling the feature map 3 times to get the output matrix $\mathbf{X}_{out}$. Each upsampling consists of an interpolation operation uses nearest neighboring interpolations for feature map up-samling, then a concatenate layer that concatenate the interpolated result and the corresponding high resolution

feature map $\mathbf{X}_{df_1}, \mathbf{X}_{df_2}, \mathbf{X}_{df_3}$ from the downsampling part. This is also called skip connection. In the last, a gated convolution layer is utilized to reduce the channel size to fit the next upsampling block. After the three times upsampling, the resolution goes back to the original size. Two standard convolution layer is added afterward without activation function to predict the surface normal. The whole upsampling branch can be represented as a regression function n,

$$n : \mathbf{X}_f, \mathbf{X}_{df_1}, \mathbf{X}_{df_2}, \mathbf{X}_{df_3} \rightarrow \mathbf{X}_{out}$$

All the convolution layers in the network use same kernel size 3×3.

One of the key feature of the network is the output has the same size as the input. This is achieved by the (1,1) padding and the same channel number in the convolution layers. Thus the surface normal can be achieved 1-1 estimation. Another key point of the network is the robustness of the noise with the help of gated convolution layers. The network can take semi-dense matrix as input then predicts the fully-dense matrix as output. The last feature is the multi-purpose using scenarios. In the description, no specific input type has been indicated. The network is also fully convoluted thus can accept different input resolutions.

3.5 Illuminated Calibrated RGB-D Image based Normal Inference

The GCNN architecture is designed for one type of input since it has only one pipe to deal with the data. In our thesis, the input is referred to structured point cloud. Thus it is suitable for geometry based approach that takes the point cloud from the input and estimate the corresponding surface normal. However, as we mentioned in the introduction, the goal of this paper is to discuss the improvement of normal inference based on illuminated calibrated RGB-D image, that is to say, we want to see if we can booster the performance of the normal estimation not only based on geometry information like point cloud or depth map, but also with illumination information, like the image and the light map. Thus we need to find an architecture to take all of these data types into account.

3.5.1 Light Map, Gray-scale Image and Vertex Map

We first introduce the required input types of the our network.

Vertex Map The vertex map $\mathbf{V}$ is converted from depth map as introduced in Chapter 4, whereas the depth map is captured from a depth camera. It contains a structured group of surface point positions in 3D space. Each pixel in the vertex map corresponding a point position. These vertices contain the geometry information of the object surface, which can be used further for training on deep learning models. For each scene, the cameras only take one shot, which consequently leaves only one vertex map. However, the vertex map is only semi-dense as we mentioned before. We take such semi-dense vertex map as input, for surface normal estimation.

Gray-Scale Image The gray-scale image $\mathbf{I}$ is captured align with the depth map based on the same calibrated camera equipment, thus the intrinsic matrix $\mathbf{K}$ and extrinsic camera matrix $[\mathbf{R}|\mathbf{t}]$ are known. Different from vertex map, it is usually fully-dense.

Light Map The light map $\mathbf{L}$ can be derived from vertex map $\mathbf{V}$ and the light source position (s_x, s_y, s_z). As shown in Figure 3.2, the incoming light direction is a vector point from light source to the surface point, therefor it can be calculated as follows

$$\mathbf{L}(x, y, z) = \frac{\mathbf{V}(x, y, z) - (s_x, s_y, s_z)}{\|\mathbf{V}(x, y, z) - (s_x, s_y, s_z)\|_2} \tag{3.5}$$

where both (s_x, s_y, s_z) and $\mathbf{V}$ are with respect to the camera space. The light direction map $\mathbf{L}$ is normalized since only the direction of the light is considered. Using the equation above for all the pixels in the point cloud can obtain the corresponding light map, which is a matrix with same size as point clouds. However, it is important to note that the light map is calcualted based on the vertex map, whereas in the training pipeline, the vertex map is only semi-dense, thus the light map is also semi-dense with the same noise feature as the vertex map, as shown in Figure 3.6.

Figure 3.6: Three kinds of input data. From left to right, vertex map, light map, gray-scale image.

3.5.2 Trip-Net Architecture

We proposed an network that takes the vertex map, light map and gray-scale image into consideration for surface normal inference, which is called Triple-pipe-gated-Network (Trip-Net). The Trip-Net employs GCNN architecture three times to accomplish the normal inference task, thus we call it "triple-pipi-gated". The architecture is shown in Figure 3.7.

The network has three pipes combined with one main pipe and two side pipes. Each pipe deals with different task. The main pipe deals with the geometry information, which takes the vertex map as the input and used to predict the surface normal. The light map input takes one side pipe and use it to extract the light feature, then forwards the features to the main pipe as a supplementary information for the normal estimation. The image map takes another side pipe to extract the image features then forward the features to the main pipe as well. The supplementary pipes provide the illumination information which helps the main pipe to refine the inferred normals.

Side-Pipe (Light) The structure of light pipe is almost identical to the GCNN architecture. It takes the light map $\mathbf{L} \in \mathbb{R}^{W \times H \times 3}$ as input, then takes it to gated convolution layers 3 times to get the the feature map, $\mathbf{X}_{L1D} \in \mathbb{R}^{W \times H \times C}$, which has the same resolution as the input map. This is the first set of feature map in the down-sampling operation. Then repeat what we did just now three times, that is for each time, 3 gated convolution layers are used to further extract the feature maps, but the different is in the last gated convolution layer, we change the stride from (1,1) to (2,2) in order to down-sampling the feature maps. Then we get $\mathbf{X}_{L2D} \in \mathbb{R}^{\frac{W}{2} \times \frac{H}{2} \times C}$,

$\mathbf{X}_{L3D} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$, $\mathbf{X}_{L4D} \in \mathbb{R}^{\frac{W}{8} \times \frac{H}{8} \times C}$ (represented as a regression function l_{down})

$$l_{down} : \mathbf{L} \rightarrow \mathbf{X}_{L1D}, \mathbf{X}_{L2D}, \mathbf{X}_{L3D}, \mathbf{X}_{L4D}$$

For the up-sampling operation, it takes 3 times up-sampling to generate higher resolution light feature maps $\mathbf{X}_{L1U} \in \mathbb{R}^{W \times H \times C}$, $\mathbf{X}_{L2U} \in \mathbb{R}^{\frac{W}{2} \times \frac{H}{2} \times C}$, $\mathbf{X}_{L3U} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$ respectively, whereas each up-sampling also considers the feature maps in the down-sampling part, (represented as a regression function $l_{up1}, l_{up2}, l_{up3}$)

$$\begin{aligned} l_{up3} &: \mathbf{X}_{L3D}, \mathbf{X}_{L4D} \rightarrow \mathbf{X}_{L3U} \\ l_{up2} &: \mathbf{X}_{L2D}, \mathbf{X}_{L3U} \rightarrow \mathbf{X}_{L2U} \\ l_{up1} &: \mathbf{X}_{L1D}, \mathbf{X}_{L2U} \rightarrow \mathbf{X}_{L1U} \end{aligned}$$

In our network, the actual up-sampling operation is done with three layers, 1, an interpolation layer based on nearest neighbor algorithm, 2, a concatenation layers to concate interpolated feature maps and the corresponding feature maps in the down-sampling part, 3, a gated convolution layer to reduce the channel size. The light pipe branch is finished in the last layer of third up-sampling.

In this pipe, we take four kinds of feature maps $\mathbf{X}_{L4D}, \mathbf{X}_{L3U}, \mathbf{X}_{L2U}, \mathbf{X}_{L1U}$ as the guided information for the further operation.

Side-Pipe (Image) The task of image pipe in the network is to predict the image feature. This pipe is a collaborate pipe with the light pipe. The architect is the same as light map pipe but only the input is the image matrix $\mathbf{I} \in \mathbb{R}^{W \times H \times 1}$. The first set of feature map is $\mathbf{X}_{I1D} \in \mathbb{R}^{W \times H \times C}$, note that it has the same channel with the light feature maps. Then down-sampling 3 times to extract the image feature maps $\mathbf{X}_{I2D} \in \mathbb{R}^{\frac{W}{2} \times \frac{H}{2} \times C}$, $\mathbf{X}_{I3D} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$, $\mathbf{X}_{I4D} \in \mathbb{R}^{\frac{W}{8} \times \frac{H}{8} \times C}$ (represented as a regression function i_{down})

$$l_{down} : \mathbf{I} \rightarrow \mathbf{X}_{I1D}, \mathbf{X}_{I2D}, \mathbf{X}_{I3D}, \mathbf{X}_{I4D}$$

For the up-sampling operation, it takes 3 times up-sampling to generate higher resolution image feature maps $\mathbf{X}_{I1U} \in \mathbb{R}^{W \times H \times C}$, $\mathbf{X}_{I2U} \in \mathbb{R}^{\frac{W}{2} \times \frac{H}{2} \times C}$, $\mathbf{X}_{I3U} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$ respectively, whereas each up-sampling also considers the feature maps in the down-sampling part, (represented as a regression function $i_{up1}, i_{up2}, i_{up3}$)

$$\begin{aligned} i_{up3} &: \mathbf{X}_{I3D}, \mathbf{X}_{I4D} \rightarrow \mathbf{X}_{I3U} \\ i_{up2} &: \mathbf{X}_{I2D}, \mathbf{X}_{I3U} \rightarrow \mathbf{X}_{I2U} \\ i_{up1} &: \mathbf{X}_{I1D}, \mathbf{X}_{I2U} \rightarrow \mathbf{X}_{I1U} \end{aligned}$$

whereas the layers in the up-sampling part has the same architecture as light pipe. In the image pipe, we take four kinds of feature maps $\mathbf{X}_{I4D}, \mathbf{X}_{I3U}, \mathbf{X}_{I2U}, \mathbf{X}_{I1U}$ as the guided information for the further operation.

Main-Pipe (Vertex) The task of vertex pipe in the network is to predict the normal map directly, which is also takes the feature maps in the other channels into account. The input is vertex map $\mathbf{V} \in \mathbb{R}^{W \times H \times 3}$ converted from point cloud. The downsampling part is still the same as the other two pipes. The feature maps in each down-sampling part has shape $\mathbf{X}_{V1D} \in \mathbb{R}^{W \times H \times C}$, $\mathbf{X}_{V2D} \in \mathbb{R}^{\frac{W}{2} \times \frac{H}{2} \times C}$, $\mathbf{X}_{V3D} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$, $\mathbf{X}_{V4D} \in \mathbb{R}^{\frac{W}{8} \times \frac{H}{8} \times C}$, respectively, (represented as a regression function i_{down})

$$v_{down} : \mathbf{V} \rightarrow \mathbf{X}_{V1D}, \mathbf{X}_{V2D}, \mathbf{X}_{V3D}, \mathbf{X}_{V4D}$$

For the up-sampling operation, the situation is different compare to the other two pipes. it fuses the corresponding resolution output feature maps altogether in three pipes to get a fused feature map $\mathbf{X}_{F3U} \in \mathbb{R}^{W\times H\times C}$, $\mathbf{X}_{F2U} \in \mathbb{R}^{W\times H\times C}$, $\mathbf{X}_{F1U} \in \mathbb{R}^{W\times H\times C}$, in each up-sampling stage, (represented as regression functions $f_{up1}, f_{up2}, f_{up3}, f_{up4}$)

$$
\begin{aligned}
f_{up4} &: \mathbf{X}_{I4D}, \mathbf{X}_{L4D}, \mathbf{X}_{V4D} \rightarrow \mathbf{X}_{F4U} \\
f_{up3} &: \mathbf{X}_{I3U}, \mathbf{X}_{L3U}, \mathbf{X}_{F4U} \rightarrow \mathbf{X}_{F3U} \\
f_{up2} &: \mathbf{X}_{I2U}, \mathbf{X}_{L2U}, \mathbf{X}_{F3U} \rightarrow \mathbf{X}_{F2U} \\
f_{up1} &: \mathbf{X}_{I1U}, \mathbf{X}_{L1U}, \mathbf{X}_{F2U} \rightarrow \mathbf{X}_{F1U}
\end{aligned}
$$

Each upsampling consists of 5 layers: 1, a interpolation layer to double size the resolution using nearest neighboring interpolation algorithm, 2, a gated layer to reduce the channel size to 1/3 of itself in order to fit the corresponding feature map in the downsampling part, 3, a concatenation layer to fuse the output with the corresponding feature map in the downsampling part, 4, a gated layer to reduce the channel size to 1/2, 5, a concatenation layer to fuse the corresponding upsamling feature map from the other two pipe altogether with the feature map in current pipe. These 5 layers consider both the information from other pipes and also keeps the high resolution from the downsampling part.

In the end of the network, in order to fit the normal inference task, two standard convolution layers are added to output normal map $\mathbf{X}_N \in \mathbb{R}^{\frac{W}{8}\times\frac{H}{8}\times 3}$ (represented as a regression function n)

$$n : \mathbf{X}_{F1U} \rightarrow \mathbf{X}_N$$

3.5.3 Loss Functions

We explored different loss function for training work of our network. The most common losses are L1 loss and L2 loss for image upsamping task. In our experiments, we used a modified BerHu Loss to train our model, which gives a better lower error in the evaluation compare to purely L1 or L2 loss.

L1 Loss L1 loss, also known as absolute error loss, which calculates the absolute difference between the prediction and the ground truth. It leads to the median of the observations.

$$L_1(\tilde{y} - y) = |\tilde{y} - y|$$

L2 Loss The standard loss function for optimization in regression problems is the L2 loss, also known as squared error loss, which minimize the squared difference between a prediction and the actual value. It leads to the mean of the observations.

$$L_2(\tilde{y} - y) = \|\tilde{y} - y\|_2^2$$

Reversed Huber Loss Inspired by [17], we use Reversed Huber loss that purposed by [24] for our model training, which indeed achieves a better error than a single L2 loss. It can be mathematically described as follows.

$$\mathcal{B}(y) = \begin{cases} |y| & |y| \leq c \\ \dfrac{y^2 + c^2}{2c} & |x| > c \end{cases} \tag{3.6}$$

where $c = 0.2 \max(|\tilde{y} - y|)$. The shape of the loss is shown in figure 3.8. The Reversed Huber loss, also called BerHu loss, is a combination with both L1 and L2 loss, where it is L1 loss in range $[-c, c]$ and L2 loss outside of this range. The loss is also first order differentiable at the connection point of two section-functions. As mentioned in [17], the parameter $c = 0.2 \max(|\tilde{y} - y|)$ corresponds the 20% of the maximal per batch error.

The chosen of these combination loss error is reasonable since it also gives more weights to the errors compare to L2 loss but also keeps the outlier suppression features in the L2 loss. The output type in our model is surface normal, which has a range between $[-1, 1]$, thus we need L2 loss to penalize larger errors, however, in the other hand, the L2 loss will suppress small values close to zero in a greater way compare to the values close to ± 1, since it is use a quadratic curve. Thus flatten the range close to zero is a good balance choice. In our experiments, the BerHu loss gives a better performance on all of the models that we have trained.

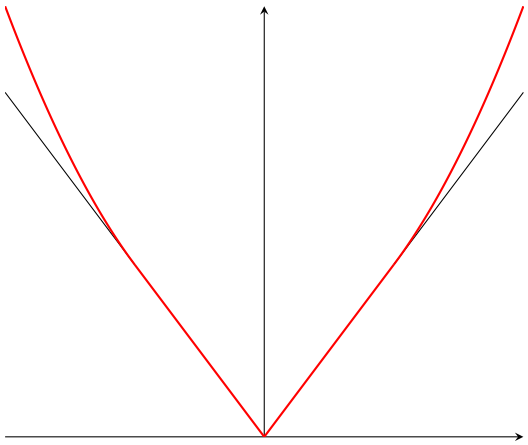


Figure 3.8: The shape of Berhu Loss (show in red line).

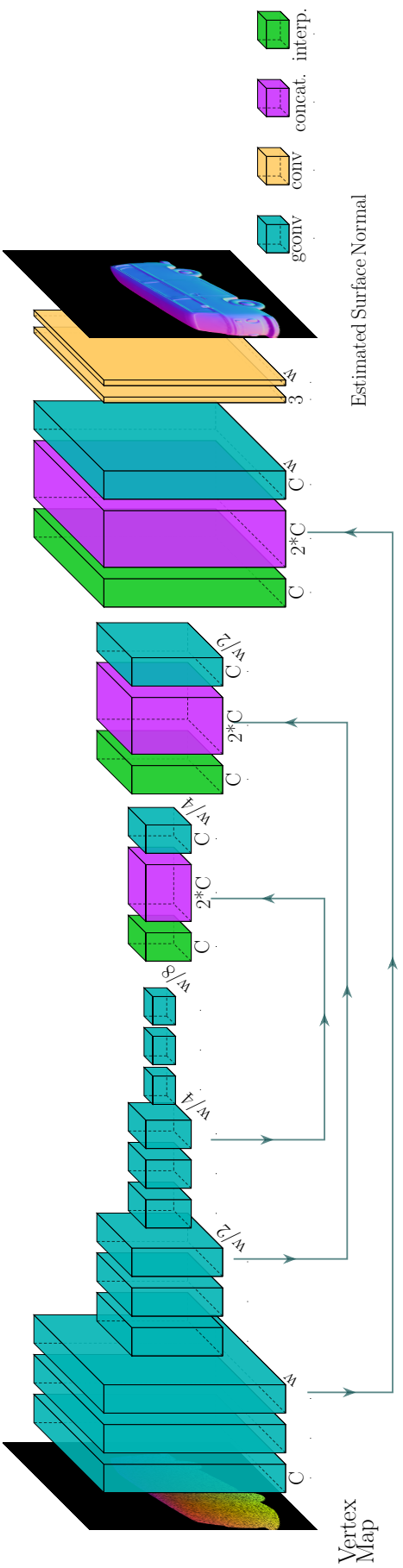


Figure 3.5: The architecture of Gated convolution neural network (GCNN) based on Gated convolution and UNet Architecture.

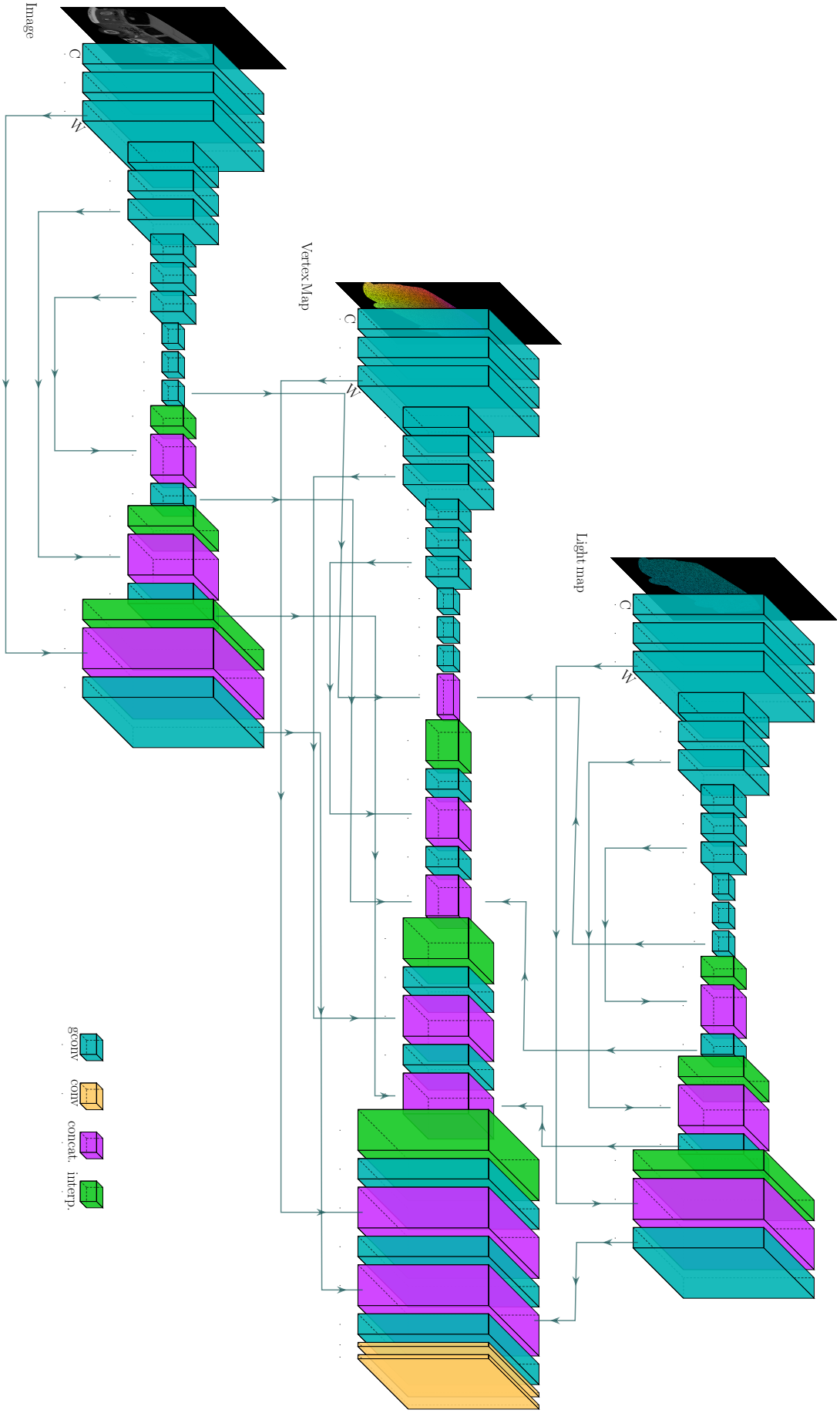


Figure 3.7: The architecture of Trig-Net

Chapter 4

Dataset

In this thesis, we require two kinds of input data to train our models. First, we need to know the geometry of the object surface. This can be collected by a depth camera, which records the Z-axis distance of the object surface to the camera position and it can be further converted to a point cloud. Second, we need illuminated information of the object surface, which considers a photometric-information used for further improvement. This kind of information requires an observed image of the object with the light directions projected onto the object surface.

In order to gain these data, we can use a light scanner to scan the objects in the laboratory. However, the scanner usually can not scan the dark, shiny, and transparent regions, which leaves plenty of missing pixels and holes in the depth map. This incomplete surface information raises the difficulty for our model training since the ground-truth corresponding to the depth map will be incomplete as well. Second, training a deep learning-based model usually requires a huge dataset, especially for a network without a backbone. To create a single depth map dataset for this thesis would be too much work and resource requirements. A proper way for getting huge depth map data with illumination information can be considered using a game engine, which can simulate any number of data that we require.

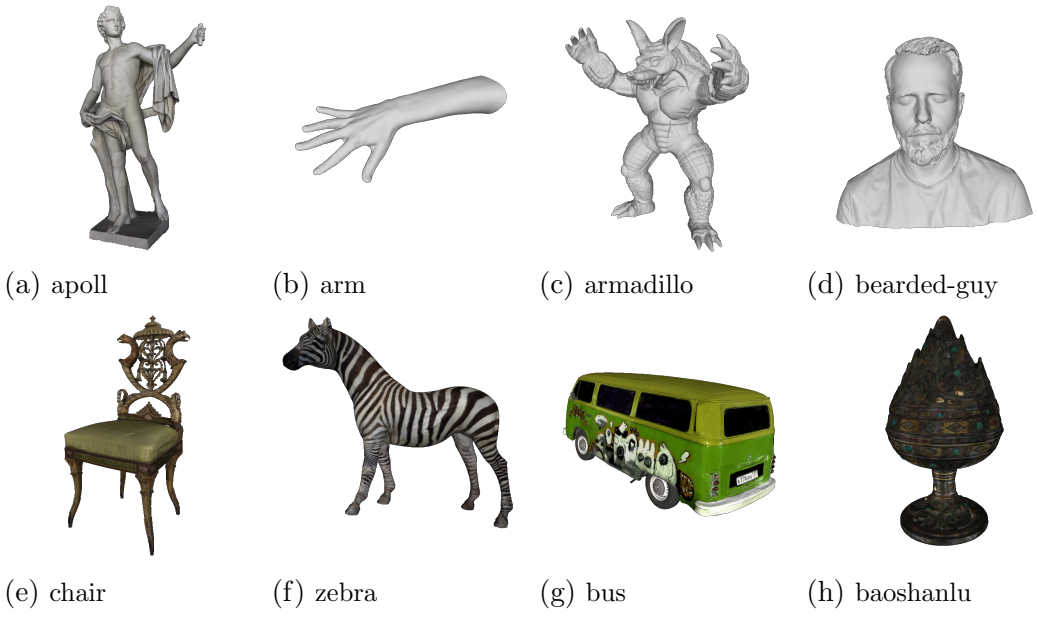
In this thesis, we use the Unity game engine for data generation. By using a *C#* script, we can set up a similar configuration in the laboratory. Then create a mount of data based on the collected objects. The dataset that we created for this thesis is called *synthetic-50-5* since it has 50 different object models for training and 5 object models for testing.

4.1 Data Resources

The object model we used are searched from internet. A set of point cloud datasets for computer vision research has been published by [31], [22], [21] and [29]. These point clouds are scanned from real objects using high resolution scanners like Cyberware 3030 MS+ and calibrated with post processing. All objects has been scanned for hundreds of times with an exhaustive completion for the origin objects, which is up to millions points. ([31]). The dense point clouds make the normal inference task trivial since the neighbor based method performs adequately for this kind of task. Some of the point clouds even contain pre-computed normal maps based on more advanced methods. They all provide the accurate ground truth for the supervised learning method.

The *synthetic50-5* is a dataset we generated in this thesis based on 50 point clouds as a training set and 5 point clouds as a test set. When we create the dataset, we tried to make the object types as many as possible to give a robust and wide range covered training scenarios. There are various categories of models, such as figure, animal, statue, toy, furniture, antique, and car model, some of which have relatively smooth

surfaces, such as /textit[arm, bus, bunny, cat, zebra], and some of which have highly complicated details, such as *Washington, Car-Engine, Armadillo*. We created this dataset for normal inference tasks. Figure 4.1 gives the illustrations of some objects. Appendix A gives a full version of dataset models.



(a) apoll (b) arm (c) armadillo (d) bearded-guy

(e) chair (f) zebra (g) bus (h) baoshanlu

Figure 4.1: Point clouds scanned by high resolution scanners

Besides, some objects also have colored textures, as shown in the second row of Figure 4.1. This is specially prepared for the illuminated approach since they will take the image into account to evaluate the surface normal. By using textured models, our models become closer to the real dataset and have improved robustness, which can apply to the real dataset further.

4.2 Synthesizing Scenes using Unity

In order to simulate the data collection scenario close to the real world as much as possible. We did the following settings. A flat cylinder called *stage* is placed in the center of the 3D space as a platform to place objects. We fixed the stage in a predetermined position, which is not changed when images are captured. To provide ambient light, one directional light with RGB color FFF4D6 /col[direction-light-color] is placed in a top-view direction at a distance of 25 meters, which also has a fixed location. An RGB-D camera is placed in a top-view direction around 10m away from the stage. The camera captures the depth map and the gray-scale image, which is randomly arranged after each scene. The moving range of the camera has 0.1m in both directions of the X, Y, and Z axis and a 1-degree random changed Euler angle. As an illumination light, a point light is placed around 5 meters away, which is also randomly moved after each scene. The moving range is 0.3m in both directions of the X, Y, and Z axis and 1 degree random in the Euler angle.

During the data collection, the object is randomly selected and placed on the stage. The object has a random up to 30 degrees rotation align *roll*-axis, 30-degrees rotation aligns *pitch*-axis, and 180-degrees rotation aligns *yaw*-axis. Thus the camera has the opportunity to capture every direction of the object and consequently abundant the

TABLE 4.1: The information saved for each scene in "synthetic50-5".

Data	**Size**
Depth map	Width×Height× 1
Depth range	MinDepth, MaxDepth
Grayscale Image	Width×Height× 1
Normal Map	Width×Height× 3
Light Position	3×1
Camera Intrinsic Matrix	3×3
Camera Extrinsic Matrix	3×4

dataset. The layout in the Unity game engine is shown in Figure 4.2. We generate 3000 scenes with resolution 128×128 and 5000 scenes with resolution 512×512.

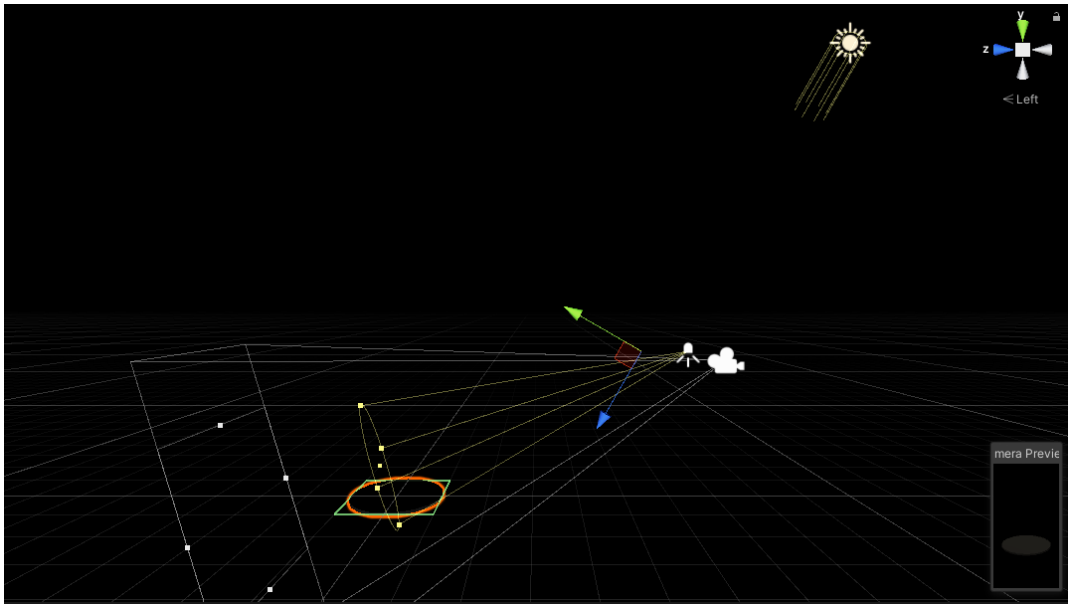
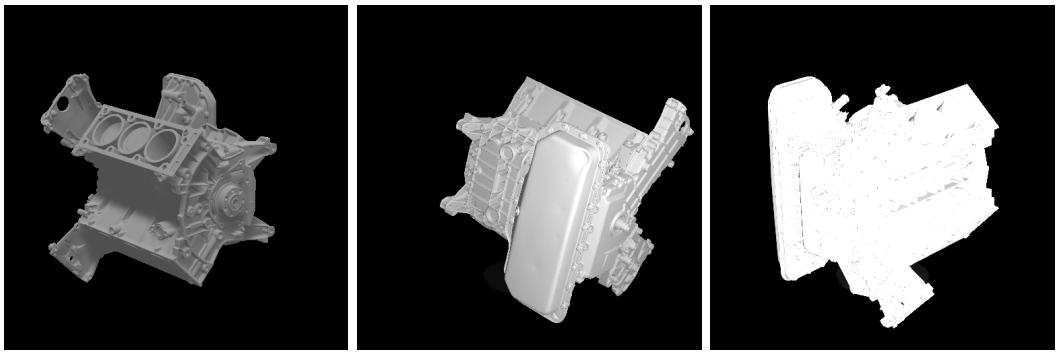


Figure 4.2: The layout of synthetic scenes generation in Unity.

The main advantage of generated scenes is the availability of complete information. We can capture the depth map in a lossless way. The corresponding normal map can also be safely considered as ground truth. And the scale of the dataset is easy to control. Table 4.1 gives more dataset information.

One critical detail that we must care about is camera exposure. Or we have to control the light radiance on the object surface. As shown in figure 4.3, too little exposure will neutralize the illumination information, resulting in an ambient light effect. Too much exposure makes the surface texture hard to recognize. When we did the experiments, we found that a suitable light setting is essential to the illumination-based approach. Too much or too few light effects won't improve the illumination-based approach.



(a) Too dim (b) Just Right (c) Too bright

Figure 4.3: Different exposure to the objects.

4.3 Data Preprocessing

The raw data captured from Unity required further preprocessing before feeding them into the training models.

Depth Map A depth map is captured by the depth camera in Unity, which is a 1 channel image that contains information relating to the Z-axis distance from object surface to the camera. It is saved as a 16-bit gray-scale image, i.e. each pixel in the range $0 - 65535$.

The raw depth maps in real data that captured by light scanners usually have missing pixels. In order to fit well with real data, the synthetic data has been added uniform distributed noise, which randomly removes the valid pixels in the maps. The simulated noise is distributed evenly around the whole map with specific noise intensity. A parameter μ is used to control the intensity of noise. It denotes the μ-percent pixel drop-off. For example, $\mu - 10$ removes 10% pixels randomly. For each scene, the noise operation based on a random μ in a range $[0, 50]$ is used. Some scenes have more missing pixels and some have fewer. The random noise intensity also enables the model to learn scenarios not only with noise but also with minor noise or even without noise. Figure 4.4 shows the noise effect with different μ.

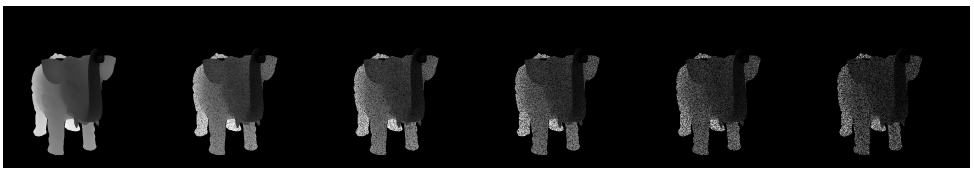


Figure 4.4: Noise-intensity on $\mu - 0, \mu - 10, \mu - 20, \mu - 30, \mu - 40, \mu - 50,$. Object Name: elephant-zun-lid.

The depth map is further converted to 3D vertex map after adding the noise.

Consider a 3-dimensional Euclidean space. Use z axis to denote the depth. The x and y axis are perpendicular to each other. For a pixel (u, v) on depth map, its depth $D(u, v)$ is the Z component of the corresponding point $P_C = (X, Y, Z)$ regarding camera coordinate system. Knowing Z, corresponding depth map pixel $D(u, v)$ and the camera focal length in pixels fk_u, fk_v, the other two component X and Y in the 3D space can be calculated based on the triangle similarity.

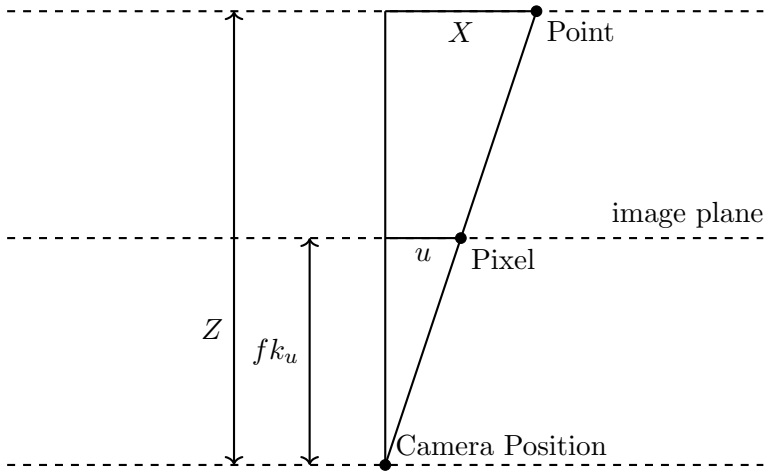


Figure 4.5: Convert depth to point in camera coordinate system

$$X = \frac{uZ}{fk_u}$$
$$Y = \frac{vZ}{fk_v}$$

Convert a point from camera coordinate system to the world coordinate system, using extrinsic matrix R and t

$$P_W = P_C R + t$$

The sizes of each training object are various, we normalized them to an unit scale thus they has relatively similar distance to the camera. We showed the value fluctuation in each axis in Figure 4.6 before normalization. Table 4.2 gives a quantitative evaluation of the corresponding average values.

The normalization has been performed as follows. First, translate the points to the original point as close as possible, then choose the range value of one axis as a scale factor, and normalize the points to unit vectors. The equation is shown as follows

$$X_n = \frac{X - \min(X)}{s}$$
$$Y_n = \frac{Y - \min(Y)}{s}$$
$$Z_n = \frac{Z - \min(Z)}{s}$$

where s is a scale factor,

$$s = \max(X) - \min(X)$$

which is calculated as the range of the X axis, but theoretically can be used by the Y or Z axes as well.

Image The gray-scale image can be used for photometric stereo approach and also as a readable information for humans. Since the image captured by the camera is in RGB format, we need to convert it to gray colors to fit our models. It is based on the following equation.

$$gray : \frac{R + 2G + B}{4}$$

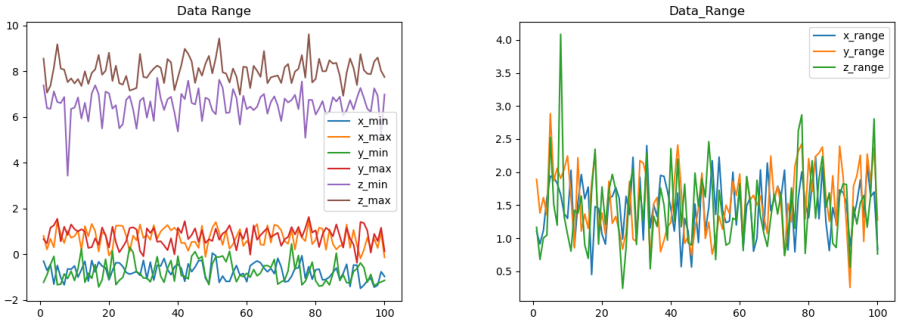


Figure 4.6: Left: Extreme value in 3 axis; Right: Vertex range in 3 axis

Axis	Scale	Min	Max
X	1.48	-0.75	0.73
Y	1.56	-0.76	0.80
Z	1.47	6.53	8.00

TABLE 4.2: The fluctuation of extreme values and their ranges in 100 random training items.

Normal Map The normal map is the tangent surface normal, which is saved in a 8-bit/channel RGB image. The surface normal (n_x, n_y, n_z) and its corresponding RGB color (R, G, B) can be converted based on the following equation:

$$n_x = \frac{R}{255} * 2 - 1$$
$$n_y = \frac{G}{255} * 2 - 1$$
$$n_z = 1 - \frac{B}{255} * 2$$

In order to save training time, we compress the dataset in PyTorch format. The structure of a single item is shown in Table 4.3.

TABLE 4.3: The structure of a single tensor in the dataset.

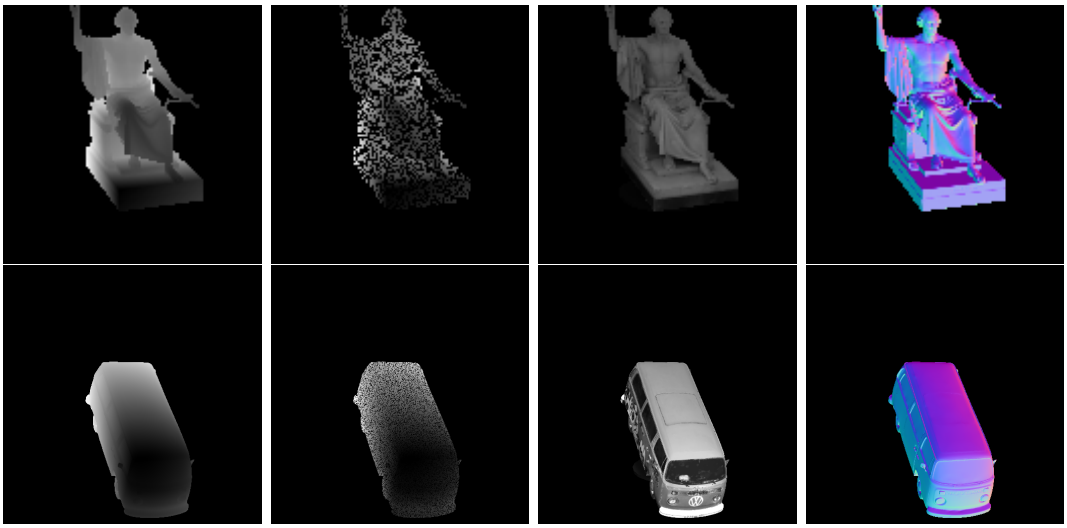
Name	Content
input-tensor	Vertex Image Light Direction
output-tensor	GT-Normal Image GT-Light-Direction
Light position	light position
Camera Matrix	K,R,t
Depth Range	minDepth, maxDepth

Chapter 5

Experiments

5.1 Training Details

The models are trained on dataset "synthetic-50-5" as mentioned in Chapter 4 with 3000 scenes. Each scene has a depth map with dimension 128×128 in height and width, an image with dimension 128×128. The depth map is converted from 3D vertex map as introduced in Chapter 4. The light map is calculated based on vertex map and the known light position. We create a tensor in PyTorch that includes vertex map, image and the light direction for each scene and considered it as one training case. Thus 3000 scenes correspond 3000 training cases. Each scene has a corresponding ground-truth normal map for loss calculation and the evaluation. Figure 5.1 shows some of the training cases. Note that the position of objects are not placed always naturally on the stage but with a random rotation in X, Y, Z axes, respectively.



(a) Depth Map (b) Add noise (c) Image (d) Normal

Figure 5.1: Some of the test scenes during the training. From top to bottom, baoshanlu, Washington, Garfield, Dragon, Bus

Besides the 3000 training scenes, 100 extra scenes containing 5 different object models are as evaluation dataset during the training. They are evaluated in each epoch.

For the training parameters, we set the training pipeline with batch size 8, we found that a higher batch size will reduce the final performance. Adam optimizer ([14]), learning rate start from 1×10^{-3}, it will decay at epoch 8 with learning decay

factor 0.5. The model is trained with PyTorch 1.10.0a0, CUDA 11.4.1, GPU with single NVIDIA GEFORCE RTXA 6000. It takes 14 hours to train GCNN and 35 hours to train the Trip-Net. We terminate the training when the angle error of the normal map stop decreasing.

GCNN model is the base model of the whole thesis. The architecture is described in 3.4. We use a single GCNN to estimate the surface normal as a geometry based approach. It uses vertex map as input to estimate the corresponding tangent surface normal map. In order to verify the applicability of the skip connection and the gated convolution layers, we trained two extra models as a comparison. The first model, we replace all of the gated layer to standard convolution layers in the network but keeps all of the other settings same, and give it a name "CNN". It is used to compare the performance between gated convolution layer and standard convolution layer for our normal inference task. As mentioned in chapter 3, the gated layer is designed to deal with noised input. Since all of the vertex map in the dataset has been added noise, the GCNN is supposed to over-perform "CNN". Another model called "NOC" is designed to verify the skip connection, which simply removes the skip connections in the network but keeps other settings same. Is is designed to show to which content will skip connection help the model performance. We use Reversed Huber Loss as loss function during the training. We found it gives a better final error compare to L2 loss.

Model	**#Total**	**V-P**	**L-P**	**I-P**	**Size /MB**
CNN	17	32	0	0	25.5
NOC	32	32	0	0	30.6
GCNN	32	32	0	0	45.8

Table 5.1: GCNN Model information. Columns V-P, L-P and I-P represent the number of convolution layers in vertex pipe, light pipe and image pipe respectively. Note that one gated convolution layer is constructed with 2 standard layers, thus it is counted as 2.

The Trip-Net model uses three times GCNN architecture with 4 times fusions, which is more difficult to train. It takes the calibrated illuminated RGB-D images as input to estimate the surface normal map. When we train this model, we take the GCNN model as a baseline, to observe the beneficial of illuminated information using with Trip-Net architecture. We also explored the optimum fusion times of the Trip-Net to see any possibility for the model simplification. A set of similar models have been trained with same settings but different fusion times, denotes by Trip-Net-Fx, where x denotes the fusion times. We evaluate the fusion times from 1 to 4. For the learning rate, we set it as $1e-3$. It goes well with GCNN model but lead to loss explosion in Trip-Net. Thus we set a learning rate schedule with an extra decay step at epoch 8. The decay factor is 0.5. The batch size is chosen as 8.

During the training we found that trip-Net with four times fusion converges obviously faster than fewer fusion times model. However, model F3 with 3 times fusion converge slower than F4 but in the end it achieves a similar evaluation loss with F4(see Qualitative evaluation). The F1 and F2 models are relatively less accurate than the other two models, but the sacrifice of accuracy gives a relatively lighter model. Since they have less fusion times, the corresponding upsampling layers in the image and light pipes can also be removed. The model can be trained in a faster way and the size is reduced as well. Table 5.2 gives a comparison of the size and training time among different models.

Model	#Total	V-P	L-P	I-P	Size /MB
Trip-Net-F1F	88	40	24	24	106
Trip-Net-F2F	92	40	26	26	137
Trip-Net-F3F	96	40	28	28	167
Trip-Net	100	40	30	30	198

TABLE 5.2: Trip-Net Model information. Columns V-P, L-P and I-P represent the number of convolution layers in vertex pipe, light pipe and image pipe respectively. Note that one gated convolution layer is constructed with 2 standard layers, thus it is counted as 2.

5.2 Quantitative Evaluation on 6 Metrics

We evaluated our models on synthetic dataset with resolution 128×128. Based on metrics proposed by [7], 6 different metrics are used for evaluation. Note that the input vertex map is only semi-dense. One of the benefit of GCNN architecture is the robustness to the noisy input, thus in the evaluation, all the points including missing points in the input vertex map are taken into account.

Average Angle Error Metric The metric calculate the average angle error for each point between the inferred normal and ground-truth normal map.

Median Angle Error Metric The metric calculate the median angle error of all the point in the normal map.

5 Degree Error Metric The metric calculate the percentage of the predicted normals that has error less than 5 degrees comparing to ground-truth.

11.5 Degree Error Metric The metric calculate the percentage of the predicted normals that has error less than 11.5 degrees comparing to ground-truth.

22.5 Degree Error Metric The metric calculate the percentage of the predicted normals that has error less than 22.5 degrees comparing to ground-truth.

30 Degree Error Metric The metric calculate the percentage of the predicted normals that has error less than 30 degrees comparing to ground-truth.

We evaluated our trained models on "synthetic-50-5". 5 objects are considered in the test dataset. They are: *Baoshanlu*, *Bus*, *Dragon*, $Garfield$, and $Washington$. Each object has 20 scenes with total 100 scenes for 5 objects. The test objects do not exist in the training dataset. We evaluate all the presented models on the test dataset, in order to fit them in one table, the name of each models are simplified. SVD model use SVD optimization method, NOC model is the no skip connection version of $GCNN$, CNN is the CNN version of $GCNN$. $F1$, $F2$, $F3$,$F4$ means the fusion times in the Trip-Net.

When evaluate the $GCNN$ models, we can take SVD model as baseline, NOC and CNN are used to verify the performance of $GCNN$ model. When evaluate the $F1 - F4$ models, we can take $GCNN$ model as baseline.

From the table we can see that all the learning based approaches achieves a better result than SVD approach. In the 30 degrees error metric, GCNN based

approach achieves 95 % accuracy whereas Trip-Net is even higher, some of the models like *dragon* achieves 98%. The best performance is around 90%, 75 %, 45 % in 22.5°, 11.5° and 5° degrees error metrics respectively. Another notable result is the close performance of F3 and F4 models, where they achieves a very comparable performance. We also found that during the training, F4 model converges faster than F3, but F3 in the end achieves a similar loss with model F4. However, based on the30°, 22.5°, 11.5° metrics, we can still see that F4 model gives a more stable performance with less high error normals. This is reasonable since the last fusion in the original resolution provides more high resolution information to the models.

Object	#	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
Baoshanlu	20	35.66	11.09	13.58	15.55	11.22	10.36	**9.77**	9.80
Bus	20	31.93	7.79	8.95	11.93	7.49	7.85	**7.30**	7.62
Dragon	20	39.57	10.60	15.29	16.03	10.47	10.23	8.16	**7.79**
Garfield	20	39.69	10.20	12.50	14.46	9.94	10.36	9.71	**9.39**
Washington	20	42.83	13.43	17.59	18.71	13.32	13.40	12.62	**12.60**

Table 5.3: Average Angle Error of the evaluation dataset.

Object	#	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
Baoshanlu	20	34.06	8.86	10.82	13.25	8.95	8.02	7.54	**7.50**
Bus	20	34.14	4.44	5.02	8.69	4.11	4.58	**3.65**	4.47
Dragon	20	36.43	7.62	11.10	13.26	7.60	7.12	5.87	**5.52**
Garfield	20	37.60	6.40	8.90	11.31	6.30	6.72	6.21	**6.04**
Washington	20	36.89	7.64	11.38	13.64	7.49	7.60	**7.03**	7.25

Table 5.4: Median Angle Error of the evaluation dataset.

Object	#	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
Baoshanlu	20	0.01	0.25	0.18	0.11	0.24	0.31	**0.33**	0.32
Bus	20	0.00	0.56	0.50	0.23	0.59	0.54	**0.63**	0.55
Dragon	20	0.00	0.31	0.17	0.10	0.31	0.34	0.43	**0.46**
Garfield	20	0.00	0.41	0.27	0.14	0.42	0.39	0.42	**0.43**
Washington	20	0.00	0.38	0.26	0.10	0.36	0.36	**0.40**	0.37

Table 5.5: Percent of error less than 5 degree of the evaluation dataset.

Object	#	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
Baoshanlu	20	0.03	0.62	0.52	0.41	0.62	0.66	**0.69**	**0.69**
Bus	20	0.05	0.81	0.78	0.65	0.83	0.82	**0.83**	**0.83**
Dragon	20	0.02	0.69	0.51	0.40	0.70	0.71	0.79	**0.81**
Garfield	20	0.03	0.72	0.62	0.51	0.73	0.71	0.74	**0.75**
Washington	20	0.02	0.62	0.50	0.40	0.63	0.62	0.64	**0.65**

Table 5.6: Percent of error less than 11.5 degree of the evaluation dataset.

Object	#	SVD	GCNN	NOC	CNN	F1	F2	F3	F4
Baoshanlu	20	0.18	0.90	0.84	0.79	0.90	0.91	**0.92**	**0.92**
Bus	20	0.26	0.93	0.91	0.89	0.93	0.93	0.93	**0.94**
Dragon	20	0.14	0.90	0.79	0.80	0.90	0.90	0.94	**0.95**
Garfield	20	0.13	0.89	0.86	0.84	0.90	0.89	**0.91**	**0.91**
Washington	20	0.14	0.81	0.72	0.72	0.81	0.81	**0.83**	**0.83**

TABLE 5.7: Percent of error less than 22.5 degree of the evaluation dataset.

Object	#	SVD	GCNN	NOC	CNN	F1	F2	F3	F4
Baoshanlu	20	0.37	0.96	0.93	0.90	0.96	0.96	**0.97**	**0.97**
Bus	20	0.43	0.96	0.94	0.93	0.96	**0.96**	**0.96**	**0.96**
Dragon	20	0.30	0.95	0.88	0.90	0.95	0.95	0.97	**0.98**
Garfield	20	0.27	0.94	0.92	0.91	0.94	0.94	0.94	**0.95**
Washington	20	0.28	0.88	0.81	0.82	0.88	0.88	**0.89**	**0.89**

TABLE 5.8: Percent of error less than 30 degree of the evaluation dataset.

5.3 Visual Evaluation on SVD

The SVD approach can predict the normal map in a good way when the given point cloud is dense. As shown in Figure 5.2. It can successfully predict the smooth surface of the dragon object, especially the flakes and the tails of the dragon.

However, it failed in the areas such as hindleg, horn and the mouth, which consists mainly by sharp edges. This is because the neighbors points in these area do not hold the assumption of coplanarity well, the normals of these neighbors can be very different. The SVD approach is depended on a well-chosen parameter k. Figure 5.3

Figure 5.2: Normal map of a dragon object predicted by neighbor based method. k=2, angle error=5. **Left**: ground-truth normal map **Middle**: predicted normal map, **Right**: Error map

shows the evaluation on different k values. When $k = 1$, the average error of the whole image is the lowest one, most of the normals are close to the ground-truth but the outline edges, which are the areas that surface normal changed extremely sever. For the case $k = 2$, the sharp edges are more smooth and cause more error, like the eyes area of the dragon. Compare to the first case, the outline edge error goes better.

Most of the edge errors are reduced when $k = 2$, since more neighbor points join the evaluation and it reduces the effect of outliers. However, for the area of horn outline, hindleg outline, the error goes worse. In this case, most of the neighbors of these points are outliers and thus failed this approach. $k = 3$ and $k = 4$ further increase the angle errors based on $k = 2$.

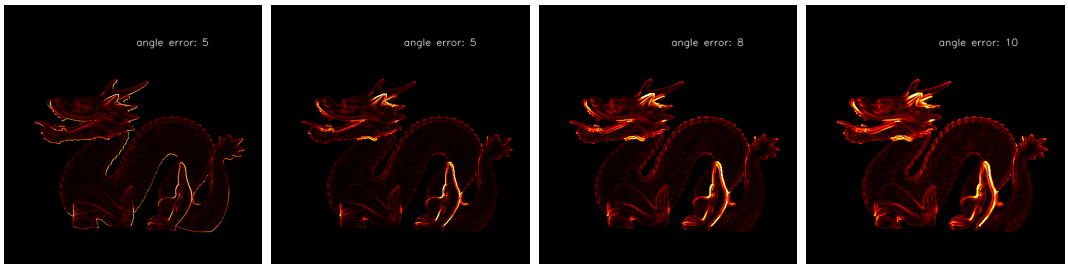


Figure 5.3: Error map of neighbor based method with different k values. From left to right, $k = 1, 2, 3, 4$ separately.

The performance of neighbor based method is good enough for a well chosen k. However, for the case of noised point cloud as input, this approach will broken, since the noise will failed the neighbor assumption and also reduce the number of possible neighbors of each point for a fixed k.

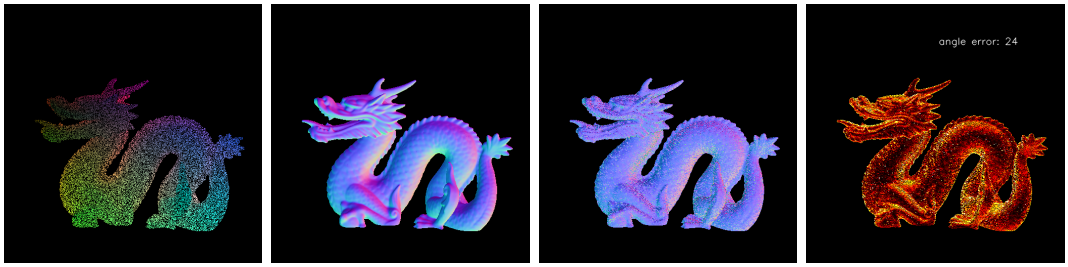
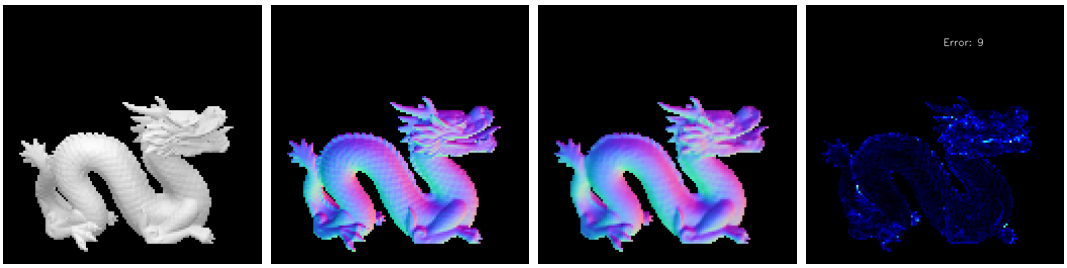


Figure 5.4: SVD visual evaluation on a noised dragon model (128×128)

5.4 Visual Evaluation on GCNN

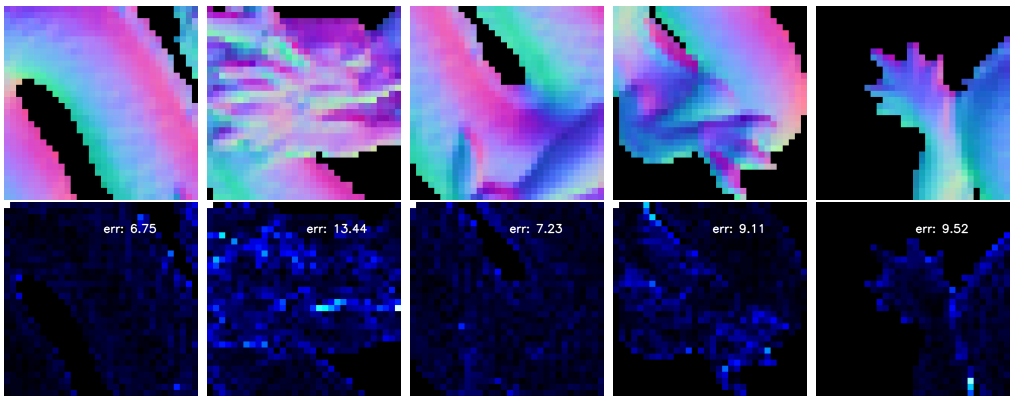
A qualitative evaluation on object "dragon" is shown in Figure 5.5. As shown in the figure, GCNN model achieved a mean angle error in 9 degrees on this dragon object. The image has an overall good performance on the whole object. A closer evaluation is shown in Figure 5.6, the normal accuracy especially good on the smooth surface, like the body area. In the same case, NOC model as shown in Figure 5.7 has a overall worse normal than GCNN model in the smooth area. CNN model keeps the skip connection thus gives a sharper result than NOC model, however, the overall smooth part of the model is still worse than GCNN. Besides, the sharp area like the hindleg and the head area of dragon object, CNN model gives a much brighter error map (which means a higher angle error). Figure B.1 shows more evaluation on GCNN model.

We can get a good result from GCNN model, but from model "Washington" we still can see it lacks the sharpness in the detail area like the face and clothes area.



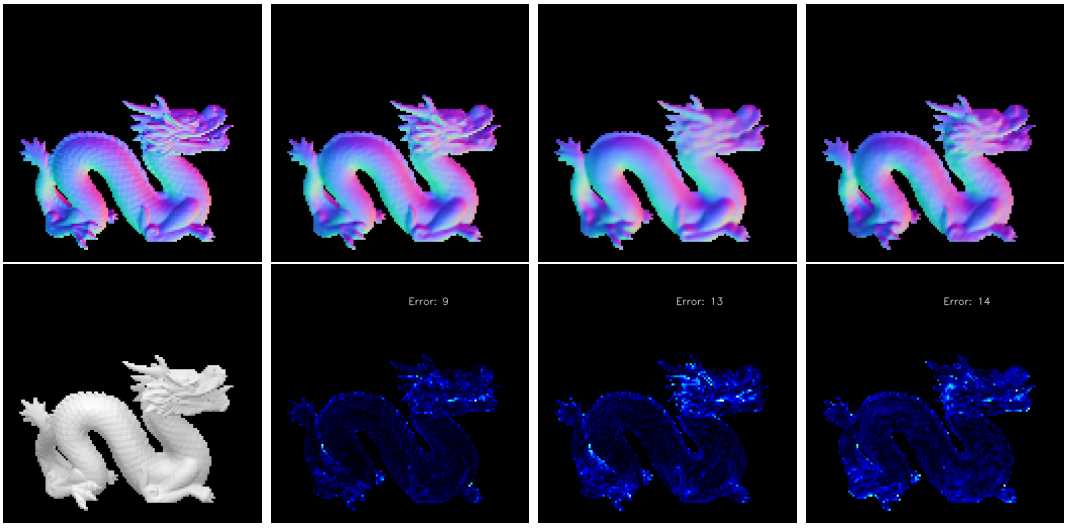
(a) Image (b) GT (c) Predict (d) Error

Figure 5.5: GCNN visual evaluation. Window size 128×128



(a) Scale (b) Head (c) Foreleg (d) Hindleg (e) Tail

Figure 5.6: Zoom in of some regions of Dragon object (GCNN models). Window size 32×32.



(a) GT (b) GCNN (c) NOC (d) CNN

Figure 5.7: Comparison of GCNN based models.

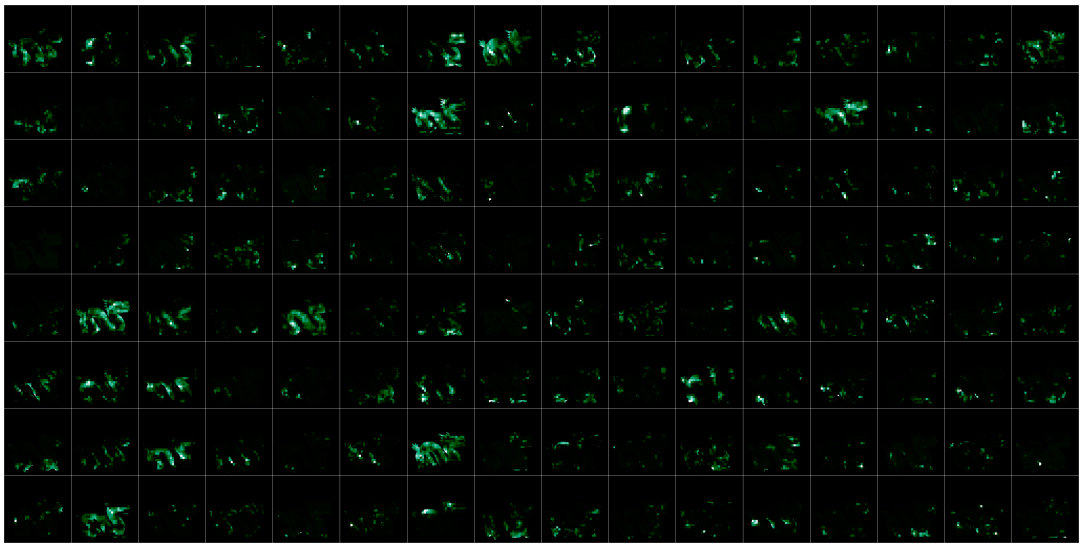
5.5 Discussion on the Feature Maps in GCNN

As shown in figure 5.8, we also visualize the feature maps of the last gated convolution layer in the first up-sampling part for CNN and GCNN models, which corresponding the 128 feature maps with size 32×32 in width and height. The input data is also same dragon object that we used for normal visualization. For each feature map, we map the minimum value to the left-most color in the color bar and the maximum value to the right-most color in the color bar. We want to discuss the the difference of the feature maps between CNN, NOC and GCNN models.

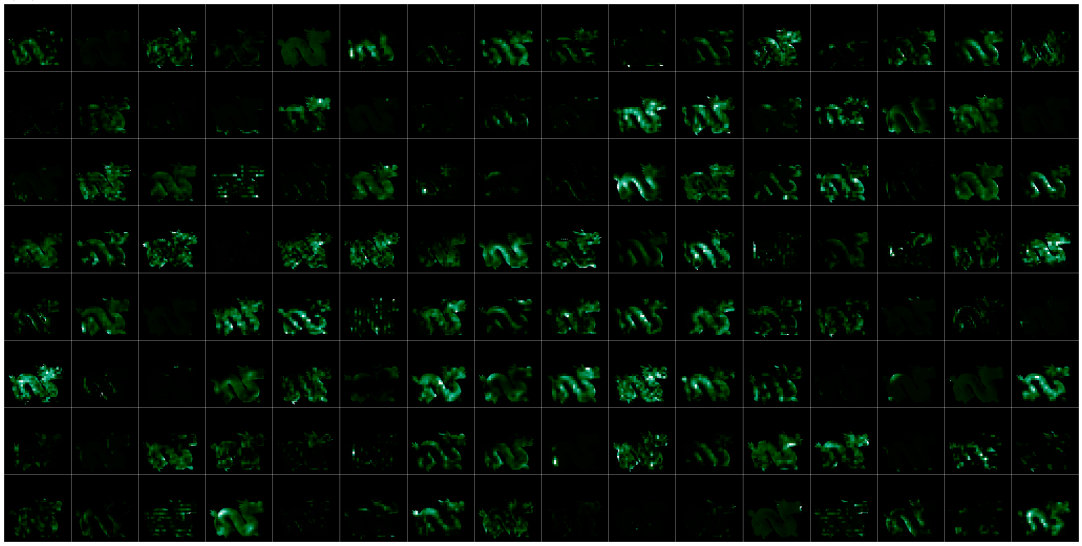
In order to see the result clearly, we use one color for the feature map visualization and use the brightness to distinguish the scale of the values, the white corresponds 1, the black corresponds 0, and the green-hue colors corresponds some values in between.

From the CNN feature maps, we can see that the high value areas are usually more than one in a single feature map. Many of them have more than two bright areas. For the GCNN feature maps, the brightness areas are usually concentrate only in one area or even one point. If we map the brightness areas to the original image, they correspond to the dense sharp edges areas, like mouse, horn, or tail of the dragon object, which is usually the hardest areas for normal estimation task. We suspect that these feature maps with few bright areas or only one bright area are the detail feature maps that for sharp edges inference. Based on this assumption, we can say that gated convolution layer describes the detail features in a more accurate way compare to standard convolution layer, since the brightness areas is smaller. Most of the GCNN feature maps usually has only one bright pixel with a few set of dim green pixels whereas CNN feature maps has more bright pixels with a lot of dim pixels all around the image. This accurate feature extraction provides our GCNN model a better performance on the detail area.

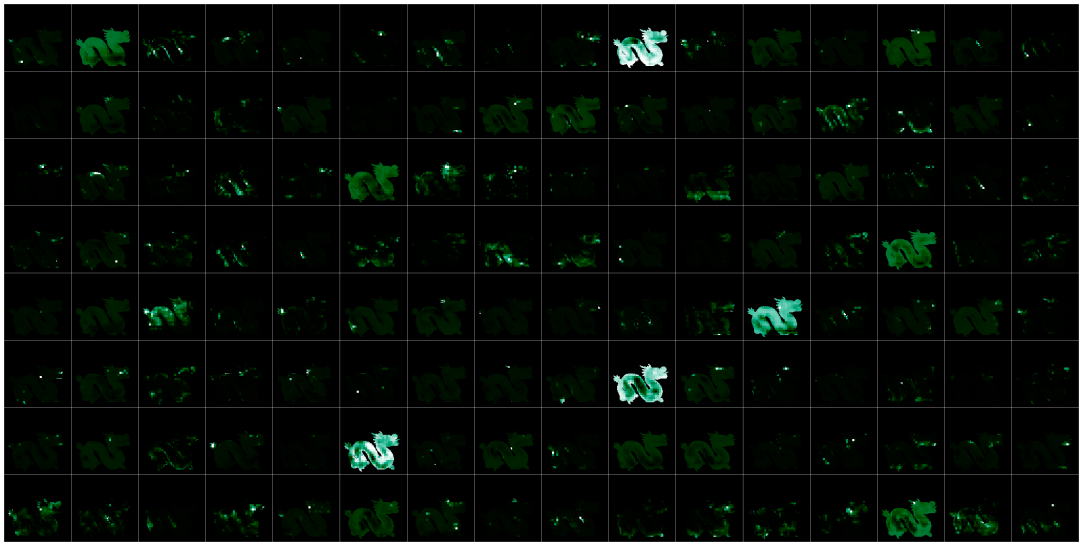
Second, for the GCNN model, we observed that some feature maps have very evenly bright areas distributed on the whole object. They are either overall dim or overall bright. Nevertheless, these feature maps are different from the detail feature maps with only few pixels with high values(brightness) but most of the other pixels with relatively low values. We suspect that these evenly value distributed features maps are the global feature maps that provide an over-view of the objects for a coarse prediction. So that they can cooperate with detail feature maps and further used to sharpen the prediction. It also holds for NOC model, since it does not equip with skip connections, most of the feature maps are global feature maps, so that the corresponding normal maps predicted by NOC model are more coarse than the GCNN model. For the CNN model, there also exists global feature maps, however, they are mostly incomplete, either missing the tails or missing the paws. Actually, these missing areas can usually be found in the detail feature maps. This also explained why the CNN detail feature maps has several bright areas in their detail feature maps. The detail feature maps and global feature maps are mixed together. And this unclear feature map extractor leads to reduce the performance on normal estimation task.



(a) CNN Feature Maps



(b) NOC Feature Maps

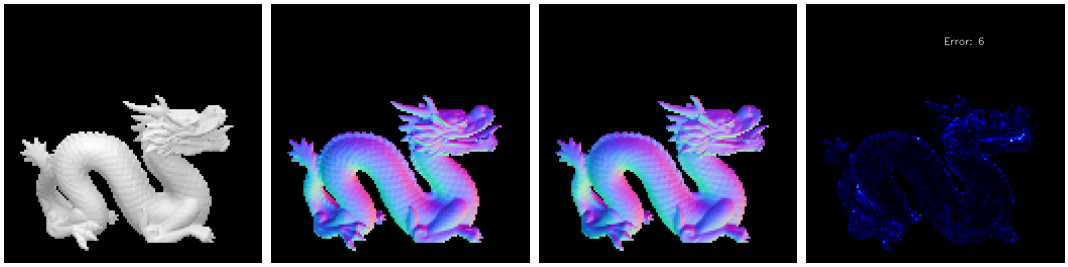


(c) GCNN Feature Maps

Figure 5.8: Feature maps of GCNN architecture based models at the first up-sampling gated convolution layer. All 128 feature maps are visualized for each model with normalization and filtered background.

5.6 Visual Evaluation on Trip-Net

For the approach using illuminated calibrated RGBD image, the task is undertaken by Trip-Net introduced in 3.5. The Trip-Net uses illuminated information align with the geometry information achieves a sharper and more accurate result compare to the GCNN model.

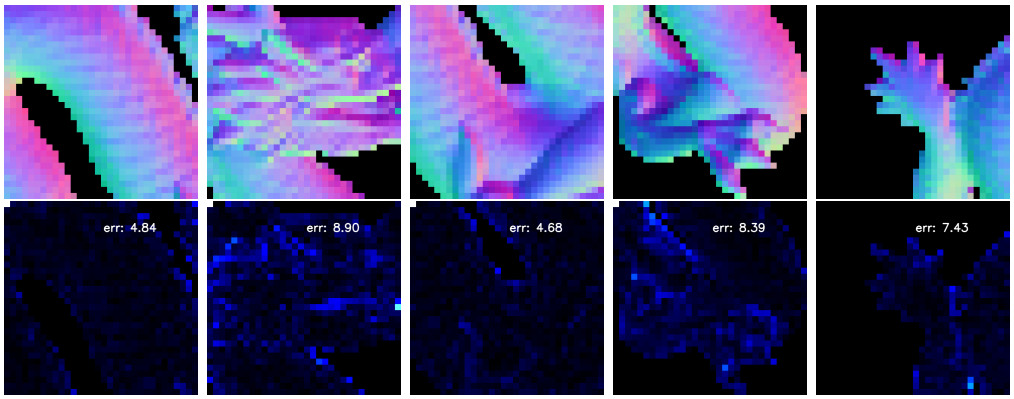


(a) Image (b) GT (c) Predict (d) Error

Error: 0 90

Figure 5.9: Trip-Net visual evaluation on resolution 128×128

The qualitative evaluation is shown in figure 5.9. In order to show the effectiveness of added illuminated information, the training settings for all the models are exact the same. We also use the same input as we did in GCNN evaluation. The error of Trip-Net is 6° whereas the GCNN is 9°.

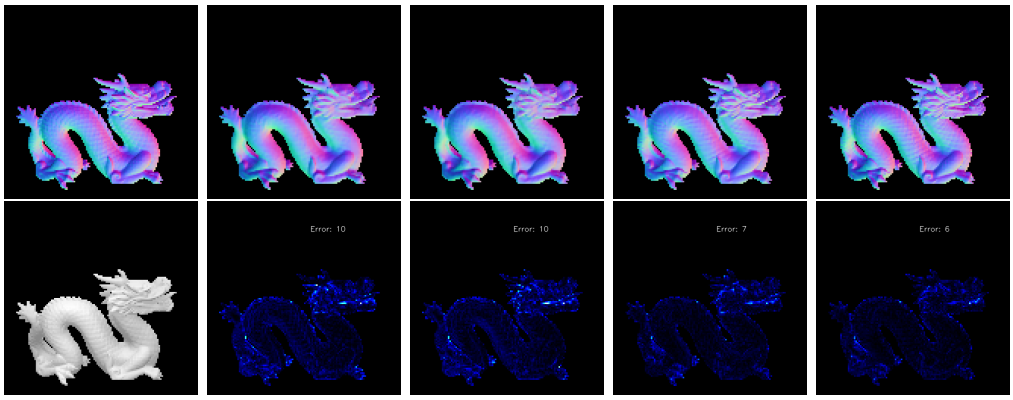


(a) Scale (b) Head (c) Foreleg (d) Hindleg (e) Tail

Figure 5.10: Zoom in of detail regions (32×32) of Dragon object (Trip-Net model).

As shown in the figure 5.10, the scale on the dragon body is much easier to detect and also close to the ground-truth. The head region gives a sharper edge prediction. All of the five sampled zoom-in regions in the Trip-Net has a better performance than GCNN.

Figure 5.11 compares different fusion times on Trip-Net model. We can see that model F1 with one fusion and and F2 with two fusions did not achieve a better result compare to GCNN model, which means the illuminated information doesn't work if we only consider the lower resolution feature maps. F3 model with 3 times fusion and F4 model with 4 times fusion give a much better result and also beyond the



(a) GT (b) F1 (c) F2 (d) F3 (e) F4

Figure 5.11: Comparison between different fusion times on Trip-Net Architecture (128×128).

GCNN model. These two times fusion include high resolution features in the last up-sampling and the final output-size features before the standard convolution layers. We can observe the dragon scale in the figures, they are much sharper in the last two images.

5.7 Trining on Higher Resolution

We also trained our models on a higher resolution dataset with 512×512 in width and height for each scene. Higher resolution data gives more information about the surface feature using the same model compare to lower resolution. The benefit is, if we extract a fixed sized patch from the normal map, say 32×32, it might correspond a hindleg area of the dragon object in 128×128 resolution but maybe only a toe in 512×512 resolution. Therefore if we still use the same kernel size in the network for the same dataset, (like we did with 3×3) the higher the resolution it has, the less area it will cover. Thus the surface in the same window size will more smooth and easy to be calculated the surface normal. This is a good thing. Because then we might only need to use these 32×32 points to calculate a toe in 512×512 resolution image. But in 128×128 resolution image, the same area 32×32 might corresponding to entire hindleg of the dragon object. In another word, we can also say that the higher resolution "smooth" the object surface within the same size area. Thus the higher resolution helps the model to calculate a more accurate normal maps.

Metrics	**SVD**	**GCNN**	**Trip-Net**
Mean	8.88	5.82	5.33
Median	3.66	3.98	3.67
5°	0.63	0.63	0.66
11.5°	0.79	0.89	0.91
22.5°	0.89	0.97	0.97
30°	0.92	0.98	0.99

TABLE 5.9: High resolution dataset evaluation of SVD, GCNN and Trip-Net models on 6 different metrics based on 100 test scenes.

Since our model is fully convolution network, the architecture keeps exactly same on the high resolution dataset. We use the same settings and the same models for training work on dataset with 512×512 resolution. The training on high resolution network takes longer time but achieves a lower angle error.

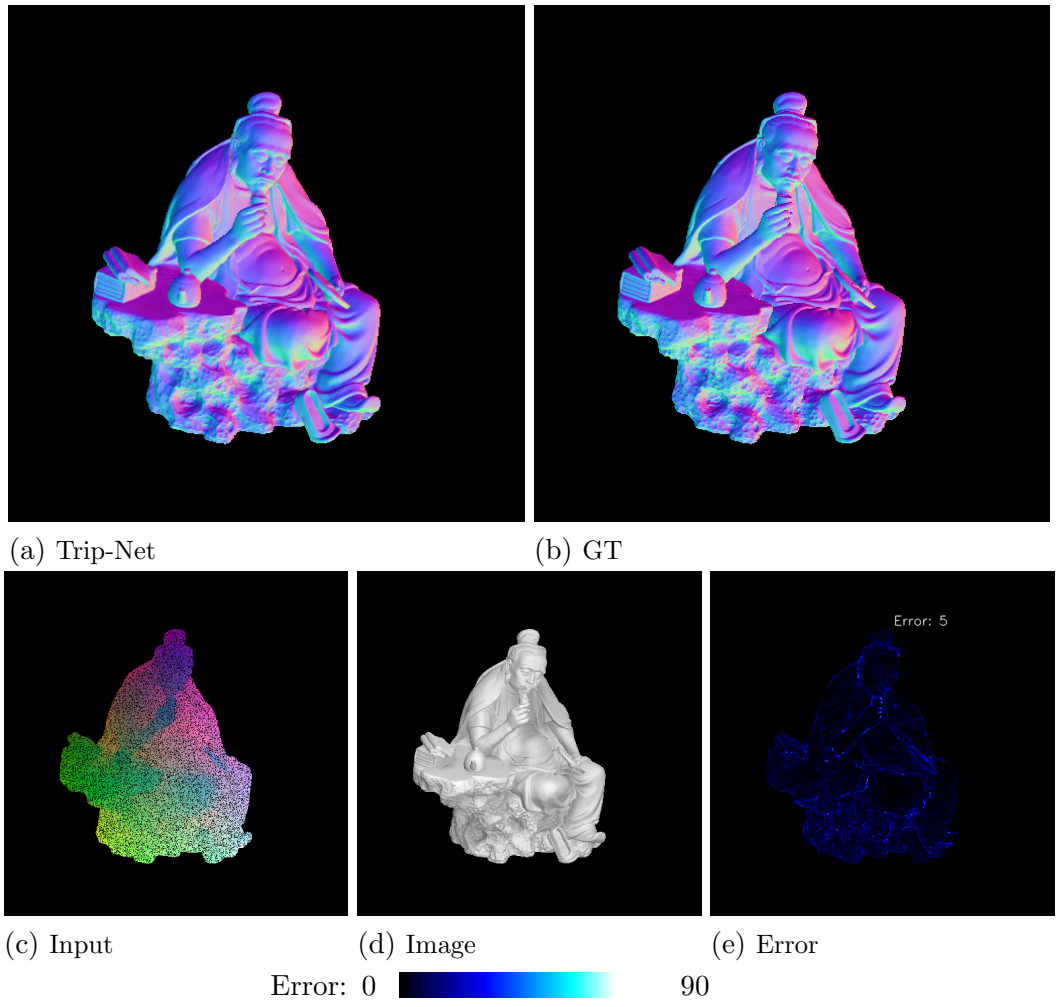
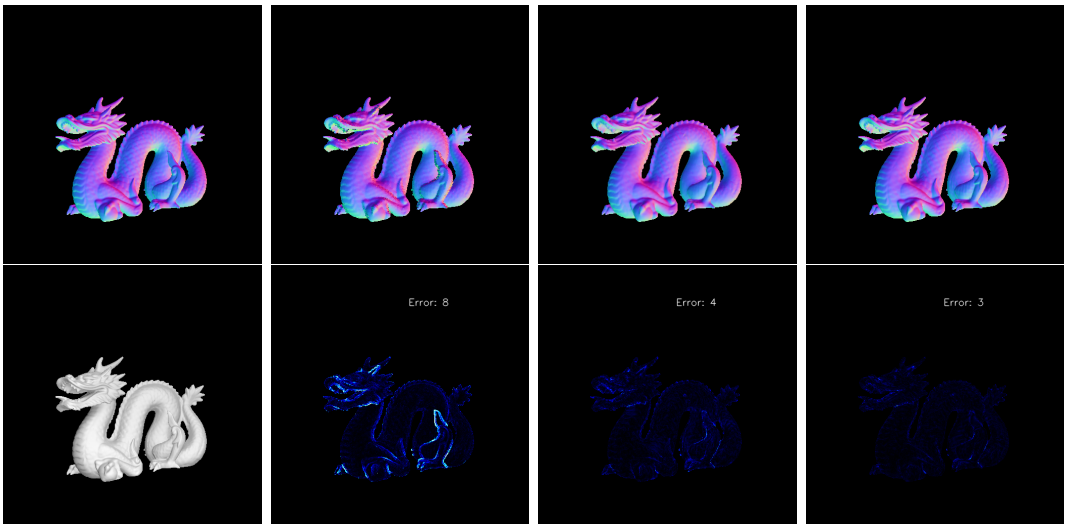


Figure 5.12: Trip-Net visual evaluation on high resolution dataset 512×512.

A quantitative evaluation is shown in table 5.9. It follows the same performance ranking compares to lower resolution, which is GCNN better than SVD approach and Trip-Net slightly better than GCNN. The accuracy is 99 % in the 30° metric. In mean error metric, the error is 5°. A qualitative evaluation is shown in figure 5.12. We use a figure object with both smooth area (the belly and arm) and highly detailed area with sharp surface (the uneven surface of stone table) for evaluation. Our models gives an 5° in mean degree metric.

A comparison with other models is shown in figure 5.13. Note that we use the same dragon object (but slightly moved the place since they have been captured in different time in high resolution dataset) for evaluation this time as a coherent comparison with low resolution dataset. The SVD based approach gives a good result (8.88° in mean degree metric). Like we discussed, the surface are relatively more smooth if we keep the same window size for normal inference. In the high resolution scenes, although the percentage of missing pixels in a fixed windows are remain the same compare to

small resolution scenes, but the remain valid pixels in the same window size are on a relatively flat plane thus they are good enough for an accurate normal inference. However, as we expected, we can still see the high degree error in the sharp edges of the dragon object, like the horns and the hindleg areas as shown in 5.13.



(a) GT (b) SVD (c) GCNN (d) Trip-Net

Figure 5.13: Comparison on high resolution dataset (512×512) .

5.8 Training on Real-Dataset

We also applied our model on a dataset that captured by a light scanner in our laboratory in order to see our model's applicability to the real-dataset. For the geometry information based approach, we directly use the GCNN model that trained on synthetic dataset since it only require the depth map as input, the scenarios of two dataset are the same. For the illuminated calibrated RGB-D image based approach, the model has to be refined based on the real-dataset, since the light intensity and position, camera matrix are different. We refined the Trip-Net based on a pre-trained GCNN model with the same settings in previous experiments, and observe the results.

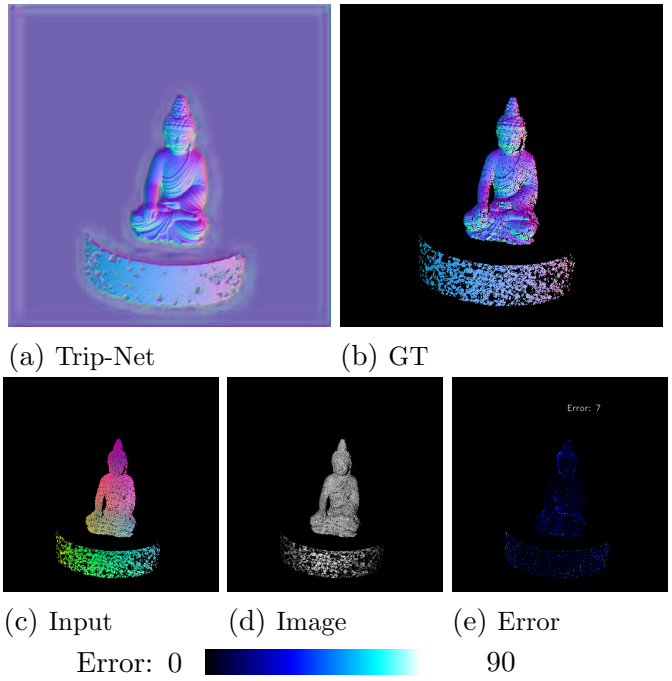
Metrics	**SVD**	**GCNN**	**Trip-Net**
Mean	8.20	8.74	**8.09**
Median	**4.87**	5.70	5.00
5°	**0.51**	0.44	0.50
11.5°	0.79	0.79	**0.81**
22.5°	0.93	0.93	**0.94**
30°	0.96	0.96	0.96

TABLE 5.10: Evaluation of SVD, GCNN and Trip-Net models on 6 different metrics based on 100 test scenes in real dataset.

Since in the real-dataset, we don't have a ground-truth for evaluation. But if we see directly from the visualization in figure 5.14, we can see that Trip-Net approach has good ability to "mend" the missing pixels in the scenes and also gives a sharp

result. The folds on the gowns are recognizable and even countable. Figure 5.15 compares the Trip-Net with other approaches.

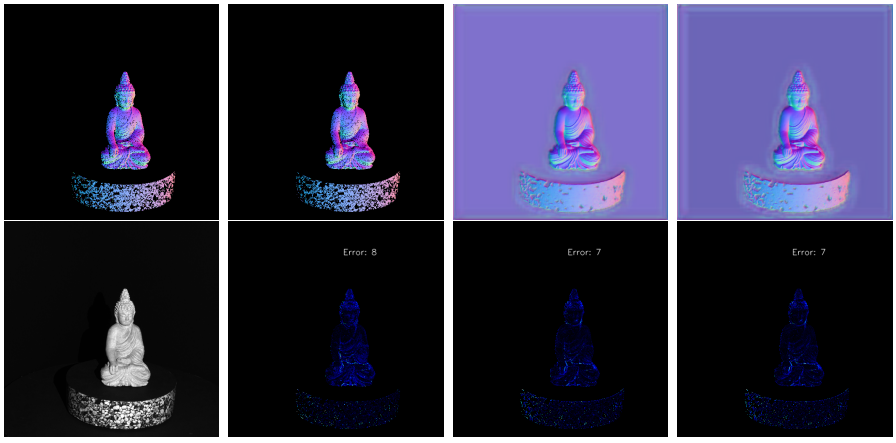
But we also noted that the big holes in the base stage remains in the normal maps. These big holes corresponds to the shinny and dark texture area, which common exist in the depth map captured by the scanners. Since these big holes are related with only special feature areas and also has irregular shapes, we didn't simulate this type of noise in the synthetic dataset but only with a sparse binary mask. Thus our models failed to mend missing big holes in the estimated normal map. This can be a further work of this topic, that find a way to generate high similar depth map noise to get a more robust training dataset.



(a) Trip-Net (b) GT

(c) Input (d) Image (e) Error

Figure 5.14: Real Dataset (512×512) evaluation (Trip-Net)



(a) GT (b) SVD (c) GCNN (d) Trip-Net

Figure 5.15: Real dataset (512×512) comparison.

Chapter 6

Conclusion

In our approaches, two networks have been proposed. The first network called GCNN is used for geometry information based approach, which is our baseline method. It is especially good for the semi-dense input data which is benefited with gated convolution layer design. Using this network we achieved a more accurate normal inference than using standard layers network based on the same architecture. We also show the effectiveness of skip connection in the U-Net. Besides, the fully convolution design enables it to use various resolution images as input.

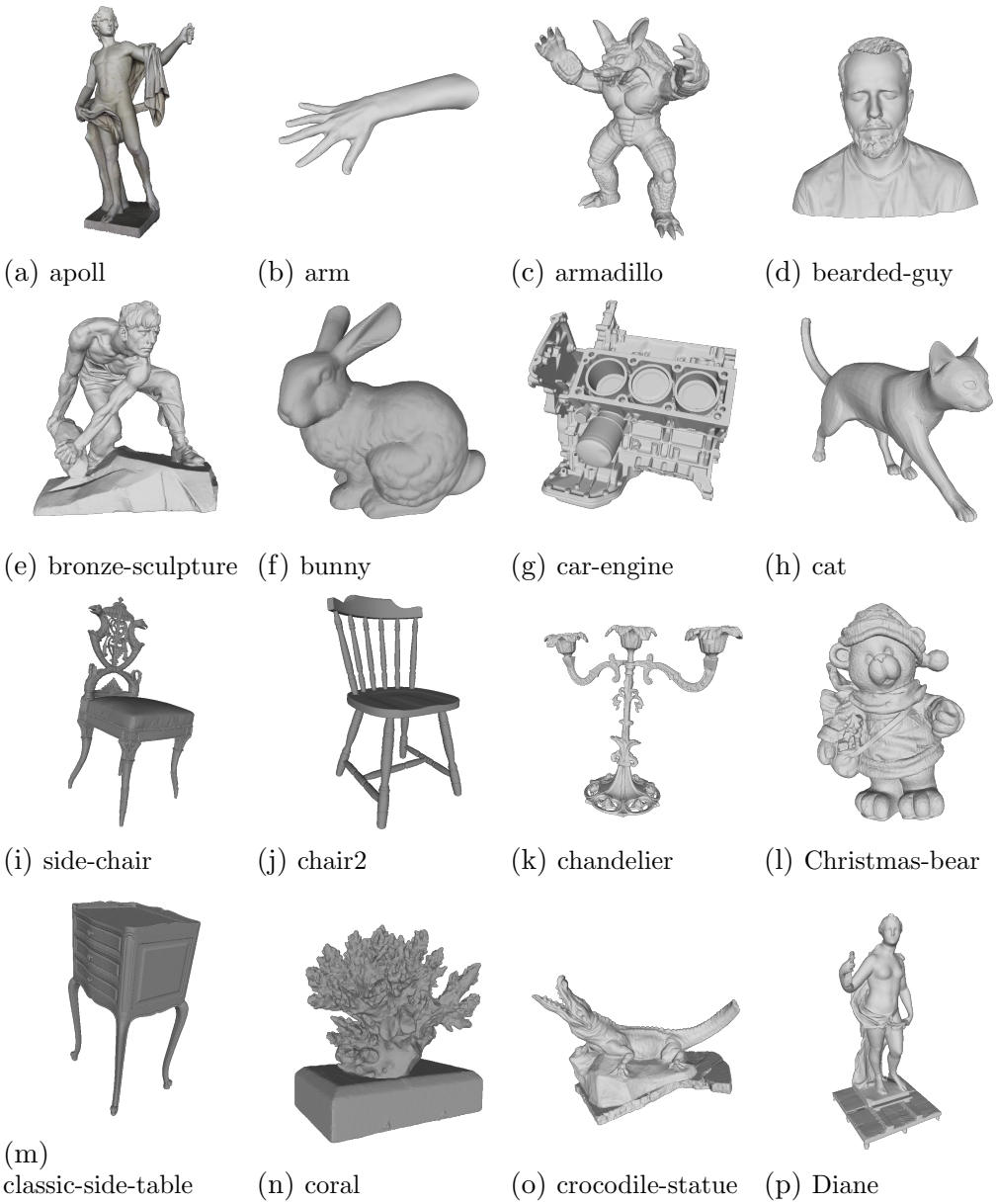
The second network is called Trip-Net, which is based on the GCNN model. It is designed for illumination calibrated RGB-D image based approaches. This model utilises GCNN architecture three times in order to handle vertex map, light map and image map separately. We have also found that 3 to 4 times fusion times among 3 pipes in our models is the key point for the performance enhancement. Based on our experiments, it shows that the models trained on illuminated calibrated RGB-D images achieve a better performance than geometry information only based approach and gives more sharpness in the normal map.

We also explored the performance of our models on a real dataset and compare with traditional method like SVD based approach. Especially, our gated convolution layer based model can fill the missing pixels in the real input data and keep an accurate normal estimation, whereas the optimization based approach like SVD is failed to do so.

We collected 55 high density point clouds from internet and used them for our dataset generation. Specifically, we use a RGB-D camera, an ambient light and one directional light collect thousands of scenes. The scene generation is simulated by game engine Unity. Furthermore, we also simulated noise in our synthetic dataset by a uniform distributed drop off to fit our model and adopt it with semi-dense input data. However, for the large missing areas, we did not generate a similar noise in the training set, which leads to our model cannot fill the holes in the real data. Exploring a sensor-noise mimic algorithm to generate more reality closed data for training work can be a direction of further work.

Appendix A

Dataset



(a) apoll (b) arm (c) armadillo (d) bearded-guy

(e) bronze-sculpture (f) bunny (g) car-engine (h) cat

(i) side-chair (j) chair2 (k) chandelier (l) Christmas-bear

(m) classic-side-table (n) coral (o) crocodile-statue (p) Diane

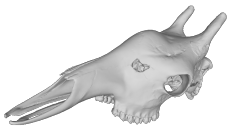
Figure A.1: Point clouds in dataset A

(a) Dresdner-knabe

(b) fantasy-dragon

(c) fox-skull

(d) Garuda and Vishnu



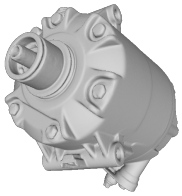
(e) giraffe-skull

(f) green-hat

(g) happy-budda

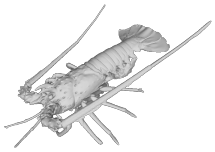
(h) heart-pendant

(i) indonesian-statue

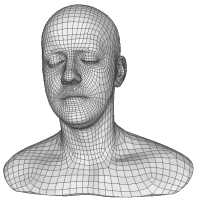


(j) industrial-compressor

(k) kylix



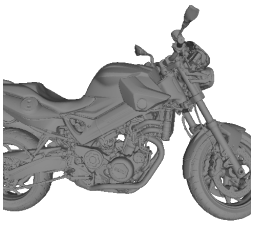
(l) lobster



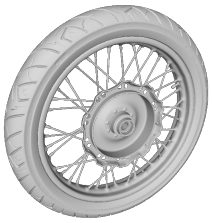
(m) head

(n) Lu-Yu

(o) Maitreya

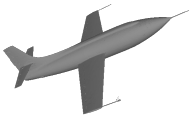


(p) motorbike



(q) motorcycle-wheel

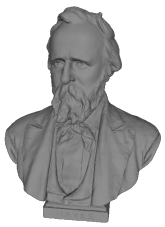
(r) Pan



(s) plane

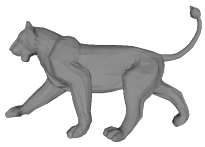
(t) elkins-refrigerator

Figure A.2: Point clouds in dataset B



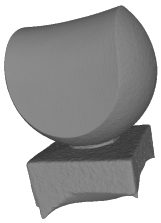
(a) Rutherford

(b) statue-dragonfly-tamer



(c) Tiger

(d) transmission



(e) universe

(f) Vestalin

(g) Zebra

(h) elephant-zun-base

(i) elephant-zun-lid

(j) cosmic-budda

(k) fangyi-base

(l) fangyi-lid

(m) guang-base

(n) guang-lid

(o) Garfield

(p) bus

(q) dragon

(r) baoshanlu

(s) Washington

Figure A.3: Point clouds in dataset C

Appendix B

More Visualization

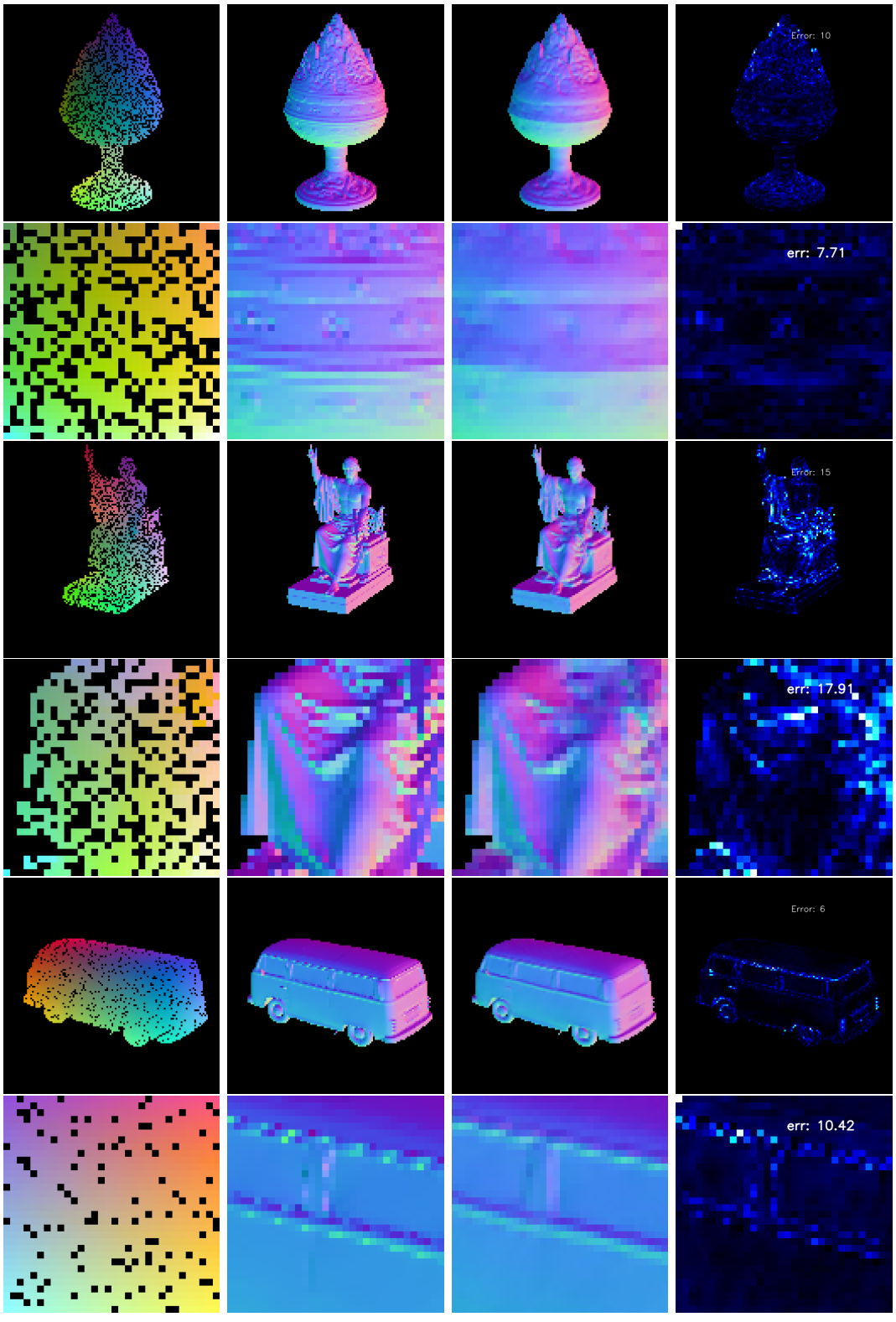


Figure B.1: GCNN evaluation on objects Baoshanlu, Washington statue, Bus(from top to bottom). (128×128)

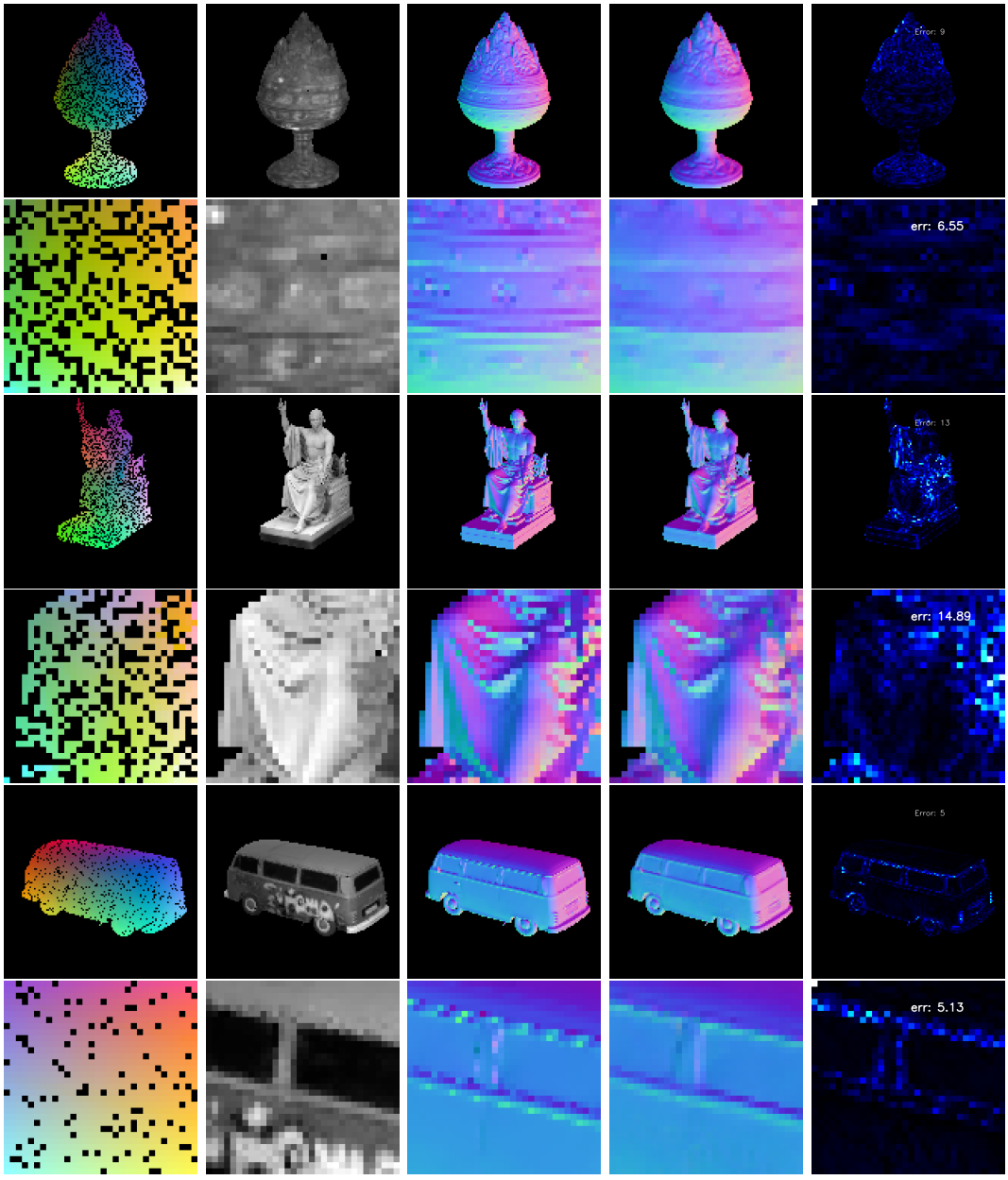


Figure B.2: Trip-Net evaluation on objects Baoshanlu, Washington statue, Bus(from top to bottom). (128×128)

List of Figures

List of Tables

Bibliography

[1] Harry Barrow and J. Tenenbaum. "Recovering Intrinsic Scene Characteristics from Images". In: *Recovering Intrinsic Scene Characteristics from Images* (Jan. 1978).

[2] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. DOI: 10.48550/ARXIV.1409.1259. URL: https://arxiv.org/abs/1409.1259.

[3] David Eigen, Christian Puhrsch, and Rob Fergus. *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network*. 2014. DOI: 10.48550/ARXIV.1406.2283. URL: https://arxiv.org/abs/1406.2283.

[4] Abdelrahman Eldesokey, Michael Felsberg, and Fahad Shahbaz Khan. "Confidence Propagation through CNNs for Guided Sparse Depth Regression". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.10 (2020), pp. 2423–2436. DOI: 10.1109/tpami.2019.2929170. URL: https://doi.org/10.1109%2Ftpami.2019.2929170.

[5] Abdelrahman Eldesokey, Michael Felsberg, and Fahad Shahbaz Khan. "Confidence Propagation through CNNs for Guided Sparse Depth Regression". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.10 (2020), pp. 2423–2436. DOI: 10.1109/tpami.2019.2929170. URL: https://doi.org/10.1109%2Ftpami.2019.2929170.

[6] Abdelrahman Eldesokey et al. *Uncertainty-Aware CNNs for Depth Completion: Uncertainty from Beginning to End*. 2020. DOI: 10.48550/ARXIV.2006.03349. URL: https://arxiv.org/abs/2006.03349.

[7] David F. Fouhey, Abhinav Gupta, and Martial Hebert. "Data-Driven 3D Primitives for Single Image Understanding". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 3392–3399. DOI: 10.1109/ICCV.2013.421.

[8] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[10] S. Holzer et al. "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 2684–2689. DOI: 10.1109/IROS.2012.6385999.

[11] Berthold Horn. "Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View". In: (Oct. 2004). URL: http://hdl.handle.net/1721.1/6885.

[12] Jiashen Hua and Xiaojin Gong. "A Normalized Convolutional Neural Network for Guided Sparse Depth Upsampling". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI'18. Stockholm, Sweden: AAAI Press, 2018, 2283–2290. ISBN: 9780999241127.

[13] Satoshi Ikehata. *CNN-PS: CNN-based Photometric Stereo for General Non-Convex Surfaces*. 2018. DOI: 10.48550/ARXIV.1808.10093. URL: https://arxiv.org/abs/1808.10093.

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[15] Klaas Klasing et al. "Comparison of surface normal estimation methods for range sensing applications". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3206–3211. DOI: 10.1109/ROBOT.2009.5152493.

[16] H. Knutsson and C.-F. Westin. "Normalized and differential convolution". In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1993, pp. 515–523. DOI: 10.1109/CVPR.1993.341081.

[17] Iro Laina et al. *Deeper Depth Prediction with Fully Convolutional Residual Networks*. 2016. DOI: 10.48550/ARXIV.1606.00373. URL: https://arxiv.org/abs/1606.00373.

[18] Zhenyu Li et al. *BinsFormer: Revisiting Adaptive Bins for Monocular Depth Estimation*. 2022. DOI: 10.48550/ARXIV.2204.00987. URL: https://arxiv.org/abs/2204.00987.

[19] Guilin Liu et al. *Image Inpainting for Irregular Holes Using Partial Convolutions*. 2018. arXiv: 1804.07723 [cs.CV].

[20] Peng Liu, Wai Lok Woo, and S.s Dlay. "One colored image based 2.5D human face reconstruction". In: (Jan. 2009).

[21] Morgan McGuire. *Artec3D*. URL: https://www.artec3d.com/3d-models/art-and-design.

[22] Morgan McGuire. *Computer Graphics Archive*. 2017. URL: https://casual-effects.com/data.

[23] Aaron van den Oord et al. *Conditional Image Generation with PixelCNN Decoders*. 2016. DOI: 10.48550/ARXIV.1606.05328. URL: https://arxiv.org/abs/1606.05328.

[24] Art Owen. "A robust hybrid of lasso and ridge regression". In: *Contemp. Math.* 443 (Jan. 2007). DOI: 10.1090/conm/443/08555.

[25] Xiaojuan Qi et al. "GeoNet: Geometric Neural Network for Joint Depth and Surface Normal Estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[26] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[28] Nathan Silberman et al. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV'12 Proceedings of the 12th European conference on Computer Vision - Volume Part V*. Springer-Verlag Berlin, 2012, pp. 746–760. ISBN: 978-3-642-33714-7. URL: https://www.microsoft.com/en-us/research/publication/indoor-segmentation-support-inference-rgbd-images/.

[29] *Smithsonian 3D Digitization*. URL: https://www.3d.si.edu/explore.

[30] Mingxing Tan, Ruoming Pang, and Quoc V. Le. "EfficientDet: Scalable and Efficient Object Detection". In: (2019). DOI: 10.48550/ARXIV.1911.09070. URL: https://arxiv.org/abs/1911.09070.

[31] *The Stanford 3D Scanning Repository*. URL: http://www.graphics.stanford.edu/data/3Dscanrep/.

[32] Jonas Uhrig et al. "Sparsity Invariant CNNs". In: *International Conference on 3D Vision (3DV)*. 2017.

[33] Robert Woodham. "Photometric Method for Determining Surface Orientation from Multiple Images". In: *Optical Engineering* 19 (Jan. 1992). DOI: 10.1117/12.7972479.

[34] Na-Eun Yang, Yong-Gon Kim, and Rae-Hong Park. "Depth hole filling using the depth distribution of neighboring regions of depth holes in the Kinect sensor". In: *2012 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2012)*. 2012, pp. 658–661. DOI: 10.1109/ICSPCC.2012.6335696.

[35] Jiahui Yu et al. *Free-Form Image Inpainting with Gated Convolution*. 2018. DOI: 10.48550/ARXIV.1806.03589. URL: https://arxiv.org/abs/1806.03589.

[36] Jin Zeng et al. "Deep Surface Normal Estimation With Hierarchical RGB-D Fusion". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 6146–6155. DOI: 10.1109/CVPR.2019.00631.

[37] Qian Zheng et al. "SPLINE-Net: Sparse Photometric Stereo Through Lighting Interpolation and Normal Estimation Networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.