Technische Universität Kaiserslautern

Master Thesis

Improved Normal Inference from Calibrated Illuminated RGBD Images

Author:
Jingyuan Sha

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

Augmented Vision
German Research Center for Artificial Intelligence

August 5, 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:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Abstract

Faculty Name
German Research Center for Artificial Intelligence

Master of Science

Improved Normal Inference from Calibrated Illuminated RGBD Images

by Jingyuan Sha

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

List of Abbreviations

I	Grayscale Image Matrix (Height × Width × 1)
L	Light Map Matrix (Height × Width × 3)
N	Normal Map Matrix (Height × Width × 3)
V	Vertex Map Matrix (Height × Width × 3)
X	Input Matrix (Height × Width × Channels)
Y	Output Matrix (Height × Width × Channels)
w	Width of Matrix
h	Height of Matrix

List of Symbols

$[AB]$	concatenation between A and B
$\oplus$	element-wise multiplication
$\cdot$	dot product

To ...

Chapter 1

Introduction

Surface normal is an important property of a surface with many applications, like surface reconstruction, shadings generation and other visual effects. However, the calculation of surface normal in many tasks is not as straightforward as simply the cross-product of two plane vectors. Especially in the task of real-world object digitalization, the surface is usually hardly to be mathematically described in equations due to the elaborate details on the objects. Instead, it is common to use a group of points to describe the object surface, which is a memory economical solution to save the fine detail of the objects and also can be easier measured by 3D scanners. Therefore the task is converted to the normal inference based on the surface point, and the most of the case, the result surface normal is merely an approximation.

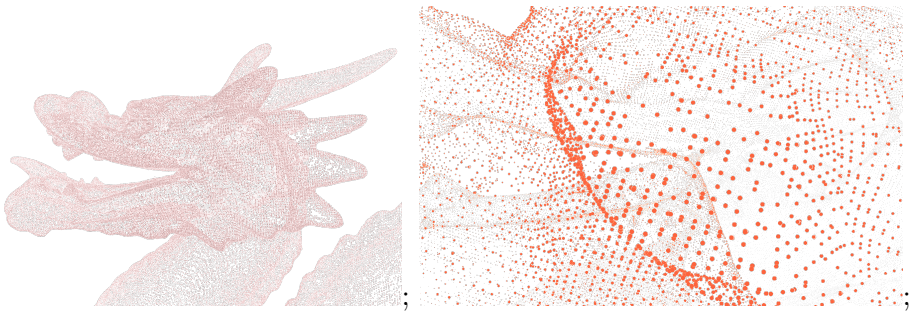


FIGURE 1.1: Left: A part of the point cloud of the dragon model. Right: The zoom in of the left point cloud.

Apart from this, due to the application scenarios, the working principle of scanners are various, which consequently produce point cloud structures in different forms. For the scanners without positions recording, the point cloud acquired after scanning is unstructured. In this case, every 3D point can be captured by different capture position, and neighbors are not defined by capture time. It increases the difficulty and computation for the neighbor based normal inference approaches. Furthermore, since the lack of inherent structure, the normal can hardly be inferred by the parallel approaches.

However, for the scanner with calibrated camera, a structured point cloud can be captured. One example is the depth camera. It captures the RGB-D images for the object, which includes the standard RGB image with depth information of each pixel. After camera calibration, a corresponding point cloud can be calculated based on the depth map and camera matrix. It gives the advantage that a structured point cloud is mapped directly based on the same capture position from a 2D depth map, the neighbor information of each point is identical to the corresponding pixel in the 2D depth map. It provides a better view for the normal inference task since the neighbor information can be considered as an important reference. Besides the

point cloud, which is one piece of information can be used for normal reference, the RGB image also leaves the trace of surface normal. If assume object surface is diffuse and the light coming from a knowing direction , the brightness of the object surface is proportional to the surface normal, which is known as Lambertian surface. It indicates the normal inference task can also utilize the relationship between surface normal and light reflection to further rectify the inferred normal map with knowing light source. Therefore, the illuminated calibrated RGB-D image provides as a good input for normal inference tasks. Unfortunately, in the actual situation, the depth maps captured by the sensors are only semi-dense, which is mainly caused by optical noises and the reflections in dark and shinny areas. Consequently, it disrupts the robustness of the normal inference methods. Median filters can be used for the sparse missing pixels, however, for the case of huge missing holes in the depth map, it produces just a paltry result. Thus a reasonable guess is required for missing areas.

Deep learning based methods provides multiple possible solutions for the challenges mentioned above. First, to deal with the noised input, deep learning based methods already have the solution for the similar tasks like image inpainting and depth density enhancement, which base on the noised image as input to predict the clean and fully dense output with or without original meanings. Therefore, it can be a help for noise in the depth map. Besides, the network of deep learning model can also handle the structured point cloud as a single input and infer the corresponding normal all together, which is very time economical comparing to the approaches like neighbor based methods. In addition, the multi-stages training architectures provide a way to consider both depth map and texture image as the input to predict the surface normals, which not only consider the point cloud but also the add the lambertian reflection as a further constraint.

In this work, we focus on the normal inference based on a semi-dense depth map with RGB image using deep learning based approaches. Specifically, the missing pixels in the depth map is filled up by the gated convolution and propagate in a customized UNet with skipping connections. The output of the training model is directly the surface normal corresponding the input depth map. The grayscale image is used to further improve the estimation accuracy. For the training work, a dataset named "synthetic50-5" is created including 55 high resolution point clouds from internet, as shown in Appendix A. The point clouds provide with the high accurate normals, which can be used as the ground truth of the training work. Most of the models have elaborate details with high curvatures but also contain smooth surfaces. The trained model is evaluated on both synthetic dataset as well as the real dataset captured from RGB-D cameras. A series of metrics have been used for qualitative evaluation. The model is shown to achieve a remarkably better prediction accuracy at a low computational cost compared to the standard approaches for semi-dense point clouds.

The structure of the thesis is as follows, Chapter 1 is the introduction of the whole work. Chapter 2 briefly discusses the related work about normal inference. Chapter 3 is the main approaches of this work. Chapter 4 introduces the created dataset for the training work. Chapter 5 is the description of the experiments and the evaluation of the models. Chapter 6 is the conclusion of the whole thesis.

Chapter 2

Related Work

Due to the different of data structures, the point clouds can be grouped into types, structured point cloud based and unstructured point cloud based. The first case contains the neighbor information of each point together with a depth map or RGB image, the second case does not contain any neighbor information which has to be further calculated.

Structured Point Cloud Based The relevant methods derive the surface normals based on spatial relationship, which utilizes the neighbor information for estimation. These methods performs well with a well-chosen window size. However, the drawbacks are that the algorithm is highly noise sensitive. It is weak in handling missing pixels, which is a common issue in the input data. The earlier methods usually use optimized methods. Holzer et al., 2012 proposed method to use local neighbors with an interest parameter p and compute the eigenvectors of the corresponding covariance matrix. They also smooth the depth data in order to handle the noise of depth image. The drawbacks are, as mentioned in the paper, the normals error go up when point depths change severely. Klasing et al., 2009 did a comparison among the optimization-based methods.

Unstructured Point Cloud Based For the unstructured point cloud, the neighbor information of each point is usually unknown. K-nearest neighbor (KNN) is a common algorithm for neighbor searching. With knowing this information, the neighbor based approaches can be used as a second step. However, KNN-method merely based on the Euclidean distance in the 3D space. Therefore, the points of the other surfaces but in a close distance will also be considered as neighbors. To ease this problem, Ben-Shabat, Lindenbaum, and Fischer, 2019 proposed a method based on unstructured point cloud with selecting the optimal scale around each point for normal estimation. To calculate the normals of multiple points in parallel Zhou et al., 2021 processes a series of overlapping patches for normal estimation.

Deep learning based methods take single RGB image or RGB-D image as the input for normal estimation. It has a strong relationship with the depth inference tasks, two benchmarks are highly used in these area. (Silberman et al., 2012 and Uhrig et al., 2017) Based on the input of training model, the methods can be roughly divided as follows:

Depth Based Depth map contains the spatial information of the object surface, which is very important for the normal inference task. However, the depth map captured by depth sensors are usually with missing pixels and holes on dark, shinny or transparent regions.(Silberman et al., 2012). To overcome the missing pixels, Yang, Kim, and Park, 2012 proposed a depth hole filling method using the depth distributions of neighboring regions. Knutsson and Westin, 1993 introduced normalized

convolution dealing with missing or uncertain data for convolution operation. It uses a binary mask to distinguish missing data and integrate it into the convolution operation. Eldesokey, Felsberg, and Khan, 2020a applied it and use normalized convolution layers in their networks, which aims to reconstruct the missing pixels from the sparse depth map sensed by cameras. Eldesokey et al., 2020 proposed an input confidence estimation network to estimate the confidence instead of using a binary mask. However, the relevant papers are only using 1 channel data as model input, which didn't discussed the case for the multiple-channel data as input, such as RGB image, or structured point cloud. Hua and Gong, 2018 further integrated RGB image as the guidance to deal with the missing pixels in the depth map.

RGB based RGB based methods predict the depth map diretly from single RGB image. Eigen, Puhrsch, and Fergus, 2014 proposed a two staged network for depth map prediction based on RGB image, which consider the global features and the local features respectively. Laina et al., 2016 employed Residual Network for the feature extraction and further designed a upsampling part which replace the fully-connected layer with the unpolling layers. Fouhey, Gupta, and Hebert, 2013 proposed a method to learn discriminative and geometrically informative primitives from RGBD images, which is further used to recover the surface normals of a scene from a single image. Qi et al., 2018 uses ResNet (He et al., 2015) to infer a coarse surface normal based on RGB image, and further refine it with the help of depth map based on the methods based on Fouhey, Gupta, and Hebert, 2013. Li et al., 2022 proposed a method achieved state of art in NYUv2 Dataset (Silberman et al., 2012), which adaptively generate information to predict depth maps based on RGB image.

RGB-D based Zeng et al., 2019 based on UNet architecture for normal estimation using RGB-D image as input. The RGB image and depth map are in the separate branches and imply a fusion module in four sections of the network to concatenate two branches. It also considers the confidence of the values in depth map.

Chapter 3

Approaches

The normal inference task estimate the surface normal of a 3D object, which is assumed that object 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 type, which use a point cloud to record the surface geometry information in the 3D space. Another data type is the images, which record the surface reflectance and shading under a light condition.

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 in this thesis. The boldface capital letters stand for matrices, whereas the boldface lowercase letters stand for vectors. The capital Greece 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 vector with dimension 3×1. We use W stands for width, H stands for height, C stands for channel.

3.2 Geometry based normal estimation

3.2.1 Approach

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

Given a structured point cloud $\mathbf{V}^{W\times H\times3}$, whereas each point in the cloud corresponding a position on an object surface. estimates 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. A normal $\mathbf{n}^{3\times1}$ at point $\mathbf{p}^{3\times1} \in \mathbf{N}$ 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 use 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 hold for the 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 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 point of any point with its neighbors, then the normal $\tilde{\mathbf{n}}$ of the tangent plane can be derived from 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, we can get the corresponding normal maps.

If we consider more than 3 neighbors, the points are usually not locate on the same plane. In this case, the equation system is over-determined. In this case, we can use optimization approach to find a plane that has the minimum distance to all the points in the 3D space. Let $\mathbf{A} \in \mathbb{R}^{(k-1)\times 3}$ denotes the vertor matrix vertically stacked by $\mathbf{v}_1, ..., \mathbf{v}_{k-1}$.

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 solution and let the normal to 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 compare to only choose 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 parameters of this approach. For the smooth surface, a relative bigger k can help the equation system to fit a plane that closer to the tangent plane. For the sharp surface area, the k has to be small enough to ensure that the points it contains 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 view point $\mathbf{s}$ 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 the simplicity, this method will be named SVD in the rest of the thesis.

3.3 Photometric Stereo

Photometric stereo was initially introduced by Woodham, 1992, which is a completely a different approach compare 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 based on the fact that the light reflected by a surface is dependent on the surface normal and the light direction.

3.3.1 Approaches

The photometric stereo actually estimates the surface normal from images. Before we introduce the 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 proposed by Barrow and Tenenbaum, 1978. It interprets the observed image into reflectance image and the shading image. As shown in Figure 3.1

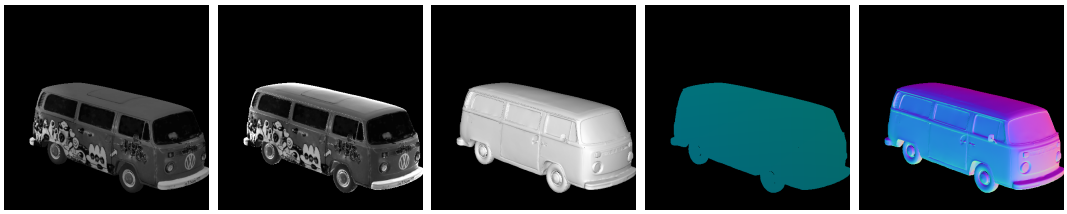


FIGURE 3.1: Intrinsic image analysis of the bus object. From left to right, original image, reflectance image, shading image, light image, normal image.

The equation can be further decomposed based on different surface models. If assume the object surfaces are Lambertian surfaces, i.e. the surface which reflect 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 the 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

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

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 image for the same scene have been captured based on different light projections. Then, for each pixel (x, y) in a scene, an equation system can be

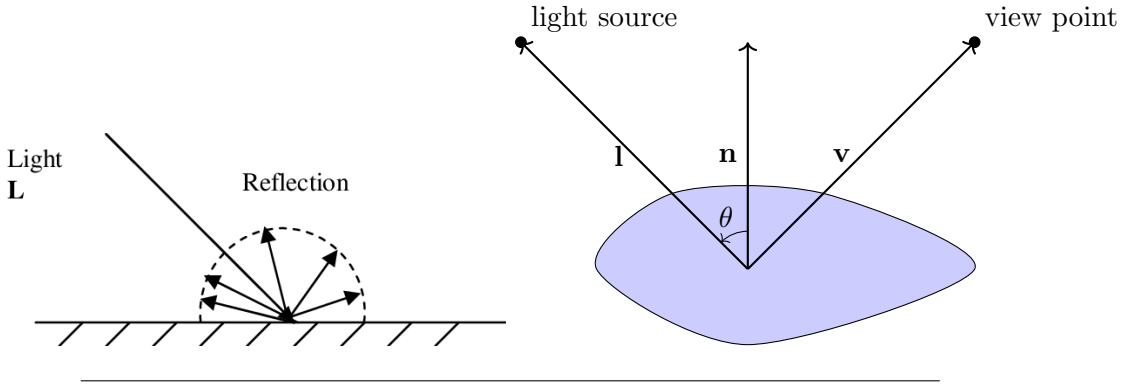


FIGURE 3.2: Left: the light reflection on a lambertian surface. (image source Liu, Woo, and Dlay, 2009). Right: The surface normal, source light direction and the view point direction, where θ denotes the angle between light direction and the normal.

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 the 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 least square methods.

First we can get the albedo with the light intensity based on the fact that normal is an 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)Horn, 2004. 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 has to be calibrated beforehand.

Shape from shading approach takes at least three light positions into account for the normal on one pixel. Thus it is 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 a great succeed for image processing. (Redmon and Farhadi, 2018, Tan, Pang, and Le, 2019) These network architectures use a batch of RGB / Grayscale images as input and are employed for classification problems. Usually, the images are convoluted with a convolution layer and down-sampling with pooling layers. The outputs of the network consist of a single value to represent the index of the corresponding class (Tan, Pang, and Le, 2019) or with a set of values to represent the position of bounding boxes.(Redmon and Farhadi, 2018). 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 as the last layers for label prediction is not suitable anymore.

Recently,Ronneberger, Fischer, and Brox, 2015 proposed an architecture called UNet for biomedical image segmentations. The architecture is shown in Figure **??**. This network has a very regular architecture for the data processing whereas each resolution level 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 use an architecture that similar like down-sampling operations, which replace the max pool layers for resize reduction to up-conv for super-resolution and remains other convolution layers. An important design is the skip connection in the up-sampling part. The feature maps that extracted in the down-sampling part will be concatenated to 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.

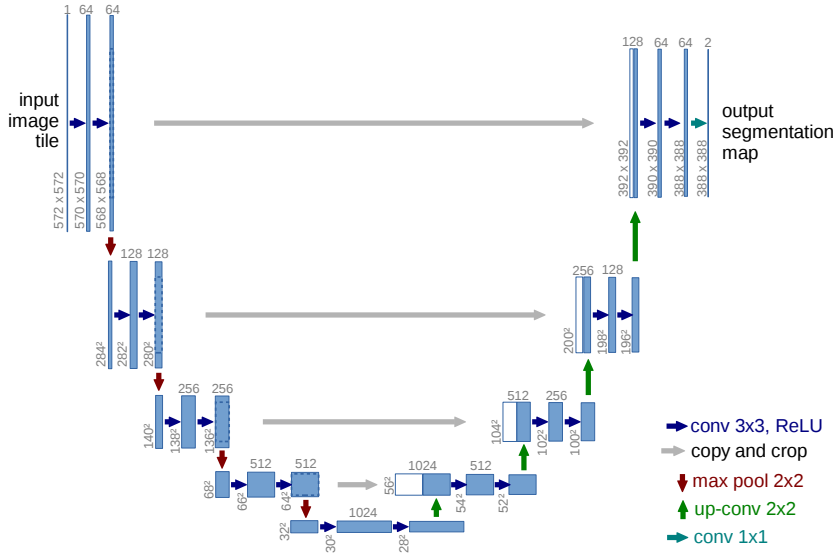


FIGURE 3.3: The structure of UNet. Ronneberger, Fischer, and Brox, 2015

The UNet is based on standard convolution layers to construct the network. This is reasonable for image processing task with full-dense input, since no missing pixels exist. However, some other input data like depth map captured from light scanner,

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 input, but then it will confused by the valid pixels and the invalid pixels (the missing pixels) during the training, and consequently leads to artifacts such as blurriness and color discrepancy, as mentioned by Liu et al., 2018. Thus it is reasonable to find a way to distinguish the valid and invalid pixels during the training.

Eldesokey, Felsberg, and Khan, 2020b use 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 kenel, M is the corresponding mask. A binary mask uses 1 to indicate valid pixels and 0 otherwise. $\odot$ denotes element-wise multiplication.

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

3.4.1 Gated Convolution

Oord et al., 2016 proposed a gated activation unit to model more complex interactions comparing to standard CNN layers, which mainly inspired by the multiplicative units exist in Long Short-Term Memory proposed by Hochreiter and Schmidhuber, 1997 and Rated Recurrent Unit (GRU) proposed by Cho et al., 2014. Yu et al., 2018 employed the same gated unit and use it as gated convolution layer for training tasks with in-complete input data, such as image inpainting task. In this gated convolution layer design, the mask using 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 $(N, C_{out}, H_{out}, W_{out})$:

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

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.

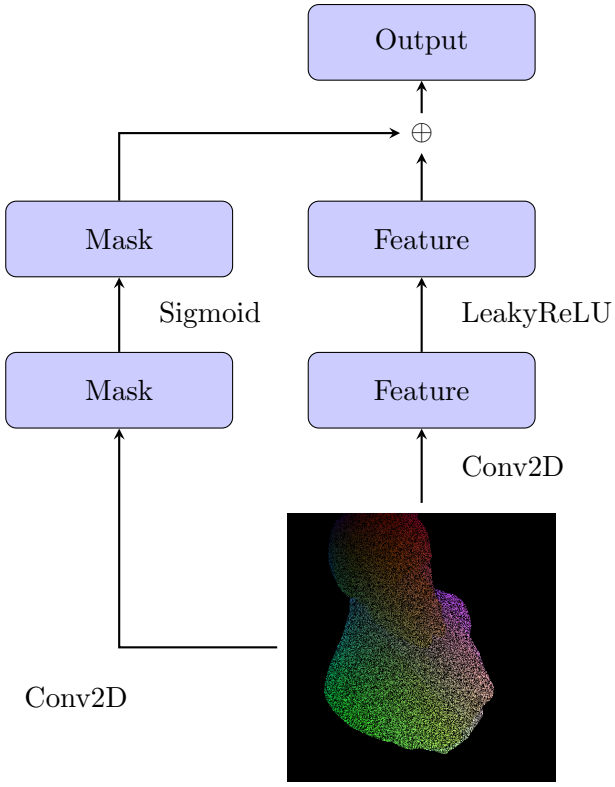


FIGURE 3.4: Gated Convolution Layer, where ⊕ denotes element-wise multiplication.

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

3.4.2 Gated Convolution Neural Network (GCNN) Architecture

Based on the implementation mentioned above, we proposed a network that based on UNet Ronneberger, Fischer, and Brox, 2015 and 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 a upsampling part. In the downsampling part, the input has shape $X_{in} \in \mathbb{R}^{H \times W \times C_{in}}$ whereas output has shape $X_{out} \in \mathbb{R}^{H \times W \times C_{out}}$ The input matrix goes through 3 times down-sampling layer block, a down-sampling block has two gated convolution layers with stride (1,1) and an extra gated convolution layers with stride (2,2) as a downsampling operation. Thus there are in total 3 gated convolution layers in each down-sampling time. The total three times downsamplings extract the geometry features X_v from input matrix X_{in} (represented as a regression function f)

$$f : X_{in} \rightarrow X_f$$

After the feature extraction, the network upsampling the feature map 3 times to get the output matrix X_{out}. Each upsampling consists of an interpolation operation uses nearest neighboring interpolations for resolution up-samling, then a concatenate layer that concatenate the interpolated result and the corresponding high resolution feature map $X_{df_1,df_2,...}$ 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,

$$n : X_v, X_{df_1,df_2,...} \rightarrow X_{out}$$

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 **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.

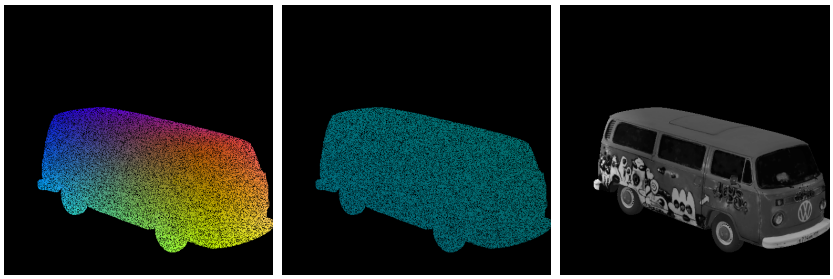


FIGURE 3.6: Three kinds of input data. From left to right, vertex map, light map, gray-scale image.

3.5.2 Trip-Net

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 Function

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 Laina et al., 2016, we use Reversed Huber loss that purposed by Owen, 2007 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 Laina et al., 2016, 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.

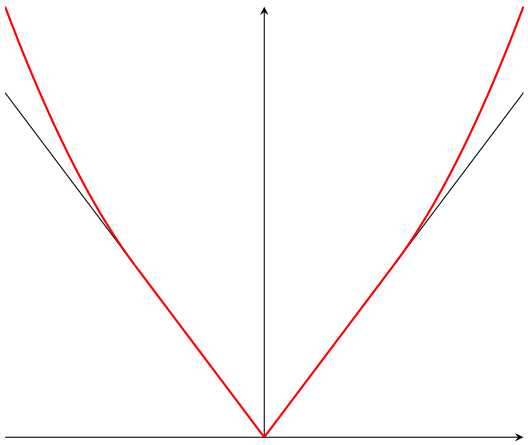


FIGURE 3.8: The shape of Berhu Loss (show in red line).

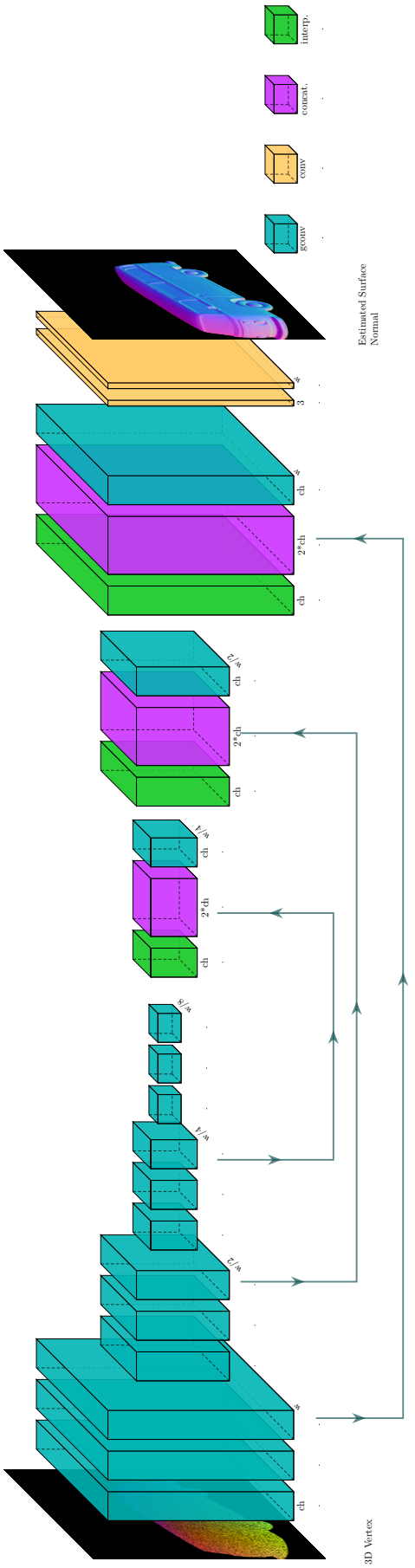


FIGURE 3.5: The architecture of Gated convolution neural network (GCNN) based on Gated convolution and UNet Architecture.

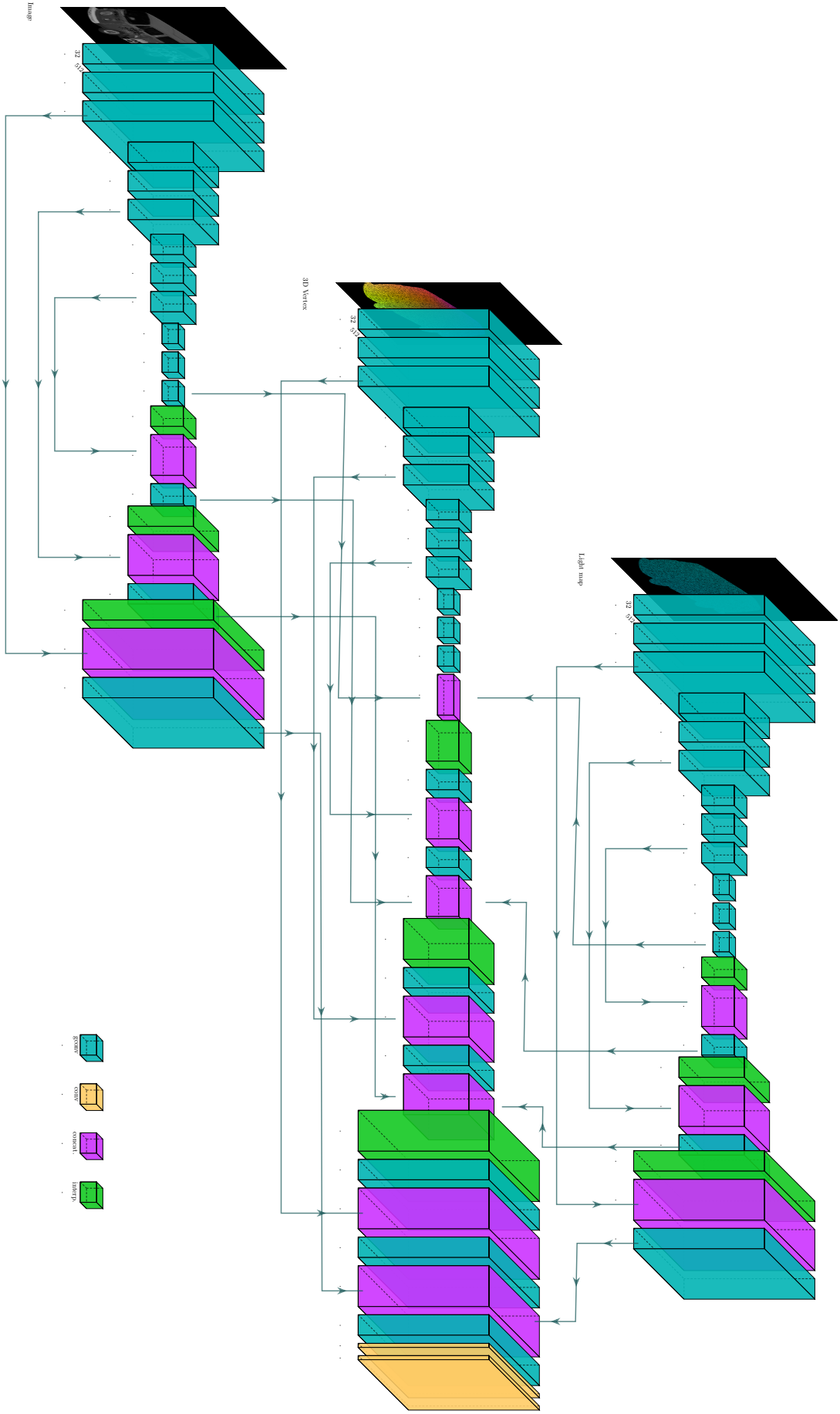


FIGURE 3.7: The architecture of Trig-Net

Chapter 4

Dataset

In this thesis, we require two kinds of input data for model training. First we need geometry information of the object surface. This can be collected by a depth camera, which records the Z-axis distance of the object surface point to the camera position. Second we need illuminated information of the object surface, which considers as a help-information of the geometry based model to further improve the inference accuracy. This kind of information is collected by a light scanner, which usually has known intrinsic and extrinsic camera matrix, so that we can convert the point between camera coordinate system and the world coordinate system. Besides, an extra spot light will be placed in a known position to illuminate the object surface. Then, the light scanner can capture the scene as a grayscale/RGB image, save the camera matrix and light position. So we can based on the light position and the depth map acquired from depth camera to calculate light direction and further use image to estimate the surface normal.

In order to acquire these data, we can use a light scanner to scan the objects in laboratory. However, the scanner is failed to scan dark, shinny and transparent region, which leaves a plenty of missing pixels and holes in the depth map. Thus the corresponding surface points are incomplete for the object. It leads to the missing ground-truth in the normal map. Second, to train a deep learning based model usually require a huge dataset, especially for the network without backbone. To create a single depth map dataset for this thesis would be too much of work and resource requirement. 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 required.

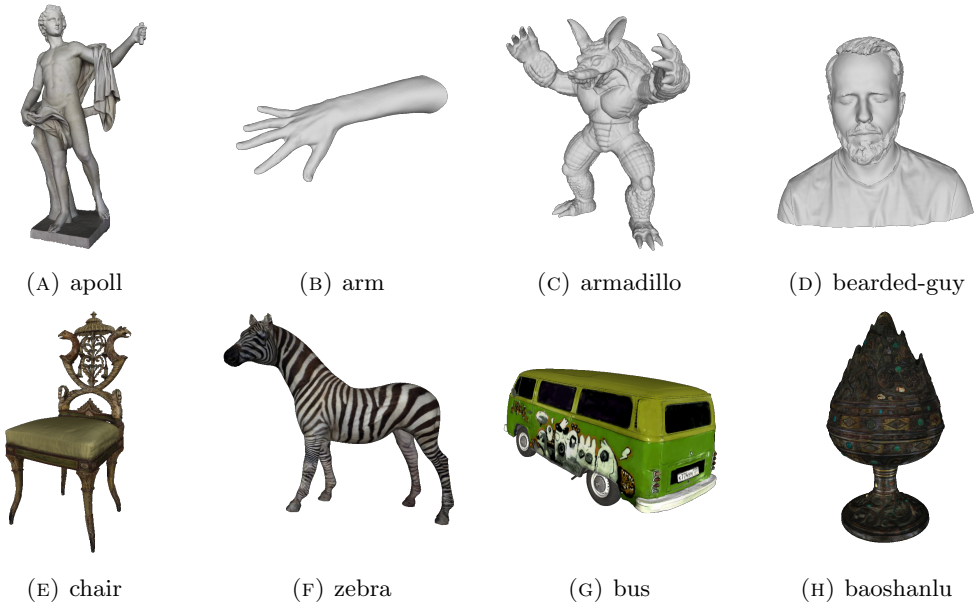
In this thesis, we use Unity game engine for data generation. Based on a *C#* script, we can set on a similar configuration in the laboratory. Then create a mount of data based on the collected object models. The dataset that we created for this thesis is called *synthetic-50-5*, since it has 50 different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

4.1 Resource

The object model we used are searched from internet. *The Stanford 3D Scanning Repository* n.d., McGuire, 2017, McGuire, n.d. and *Smithsonian 3D Digitization* n.d. published a set of point cloud dataset for computer vision research. These point clouds are scanned from real objects using high resolution scanners like Cyberware 3030 MS+ and calibrated with post processing. Each objects has been scanned for hundreds of times with an exhaustive completion for the origin objects, which is up to millions points. (*The Stanford 3D Scanning Repository* n.d.). The dense point clouds makes the normal inference task trivial since the neighbor based method performs good enough for these kind of task. Some of the point cloud even equipped with

pre-computed normal map based on more advanced methods. They all provides the accurate ground-truth for the supervised learning method.

The *synthetic50-5* is a datset we generated in this thesis with 50 point clouds as training set and 5 point clouds as 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. The category of the models include figure, animal, statue, toy, furniture, antique, car model, which include the smooth surface model like *arm, bus, bunny, cat, zebra* and also the model with highly elaborate details, like *Washington, Car-Engine, Armadillo.* The dataset is created base on the work of this thesis and using for normal inference task. Figure **??** 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 of the objects also has its own texture, as shown in the second row of Figure **??**. This is specially prepared for illuminated approach since they will take the image as account to evaluate the surface normal. The object surface gives a different reflectence with different surface texture, which further impact the model performance.

4.2 Synthesize Scenes using Unity

In order to simulate the the data collection scenario close to the real life in the laboratory as much as possible. We placed a flat cylinder called *stage* in the center in the 3D space as a platform to place objects. The stage is fixed in a determined position, which is not moved when the scene is captured. A directional light with RGB color FFF4D6 placed in a top-view direction around 25 meters away to provide as ambient light, which also has a fixed position. A RGB-D camera is placed in a top-view direction around 10m away from the stage. The camera is the sensor to capture the depth map and the grayscale image, which is random moved after a new scene. The moving range of the camera has 0.1m in both directions of X, Y, and Z axis and 1 degree random changed Euler angle. A point light placed around 5 meters

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

away as the illumination light, which is also randomly moved after a new scene. The moving range is 0.3m in both directions of X, Y, and Z axis and 1 degree random in Euler angle.

During the data collection, the object is randomly selected and placed on the stage. The object has a random upto 30 degrees rotation align *roll*-axis, 30-degrees rotation align *pitch*-axis and 180-degrees align *yaw*-axis. Thus the camera has the opportunity to capture every direction of the object and consequently abundant the dataset. The layout in the Unity game engine is shown in Figure 4.2. We generate 3000 scenes with resolution $128 \times 128 \times 3$ and 5000 scenes with resolution $512 \times 512 \times 3$.

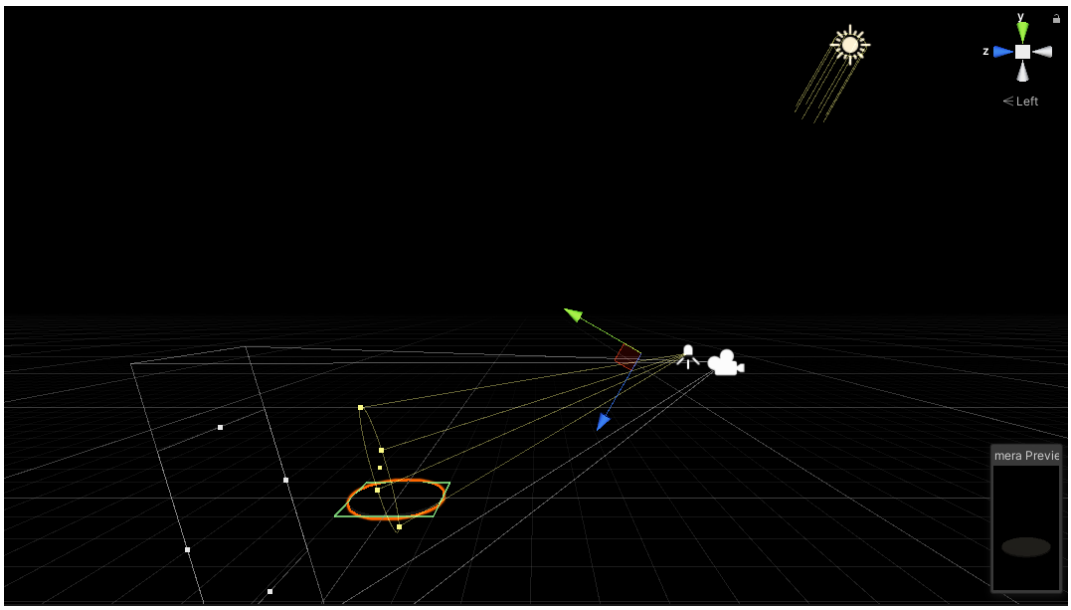
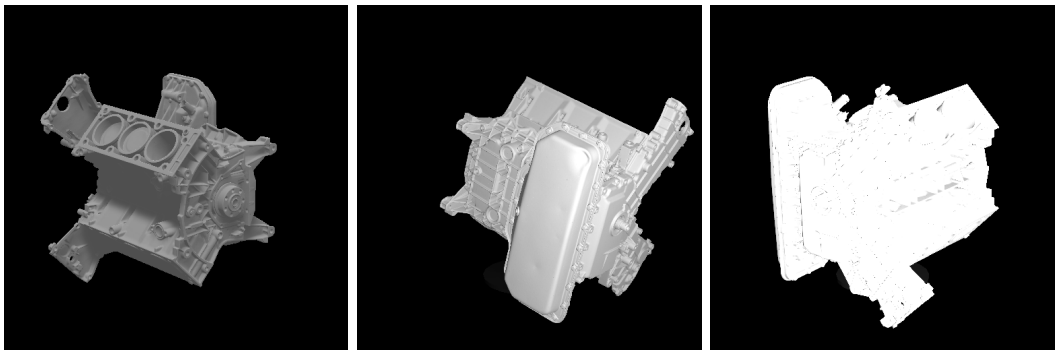


FIGURE 4.2: The layout of synthetic scenes generation in Unity.

The main advantage using generated scenes is the availability of complete information. The depth map can be captured in a loss-free 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 important detail that we must care about is the camera exposure. Or we have to control the light radiance on the object surface. As shown in figure 4.3, too less exposure on the object neutralizes the illumination information, which only left 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 illuminated based approach. A too much or too less light effects won't improve the illuminated 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 feed them into the training models. A depth map is captured by a depth camera in Unity, which is a 1 channel image that contains the information relating to the distance of the surfaces of the scene objects from a viewpoint. It can be saved as a 16-bit grayscale image, i.e. each pixel in range $0 - 65535$.

The grayscale image can be used as guided normal inference task and also as a readable scene for human. Since the image captured by camera is RGB format, we need to convert it to gray-color for our models. It based on following equation

$$gray : \frac{r + 2g + b}{4}$$

The normal map is the tangent surface normal, which is saved in 32-bit RGB color image. The surface normal (n_x, n_y, n_z) and its corresponding RGB color (R, G, B) can be converted based on following equation:

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

If consider the relation between Lambertian reflection and normal direction, the light source can be used to calculate the reflect direction of each point.

The camera intrinsic and extrinsic matrix helps point cloud calculation.

4.3.1 Convert to Point Cloud

The depth map can be converted to 3D vertex point cloud as the input of the normal inference model.

Consider a 3-dimensional Euclidean space. Use z axis denotes the depth. The x and y axis perpendicular with each other. For a pixel (u, v) on depth map, its depth

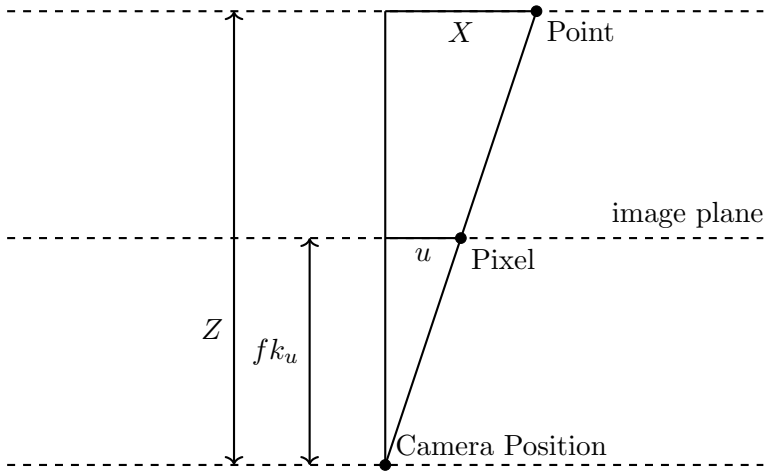


FIGURE 4.4: Convert depth to point in camera coordinate system

$D(u, v)$ is the Z component of the corresponding point $P_C = (X, Y, Z)$ with respect to camera coordinate system. The X and Y can be calculated based on the triangle similarity

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

where fk_u, fk_v is the focal length in pixels align u and v axes. Converted a point from camera coordinate system to world coordinate system, using extrinsic matrix R and t

$$P_W = P_C R + t$$

4.3.2 Point Cloud Normalization

The sizes of each training object are various, whereas it should be as an invariant value for the training model. Thus the normalization is required before feed training objects into the models. The data fluctuation in each axis is shown in Figure **??**. Table **??** gives a quantitative measurement of 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, 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,

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

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

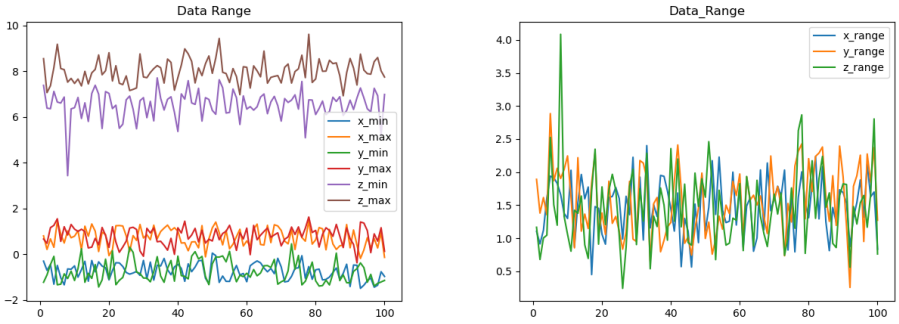


FIGURE 4.5: Left: Extreme value in 3 axis; Right: Vertex range in 3 axis

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.

4.3.3 Noise

The raw depth maps captured by light scanner usually have missing pixels. In order to fit well with real-data, the synthetic data has been added uniform distributed noise, which is randomly remove the valid pixels in the maps. The simulated noise is distributed evenly around the whole map with a specific noise intensity. A parameter μ is used to control the intensity of noise, it denotes the μ-percent pixel dropoff. For example, $\mu - 10$ removes 10% pixels randomly. For each scene, the noising operation based on a random μ in a range $[0, 50]$ is used. Some scenes have more missing pixels and some have less. 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 4.6 shows the noise effect with different μ.

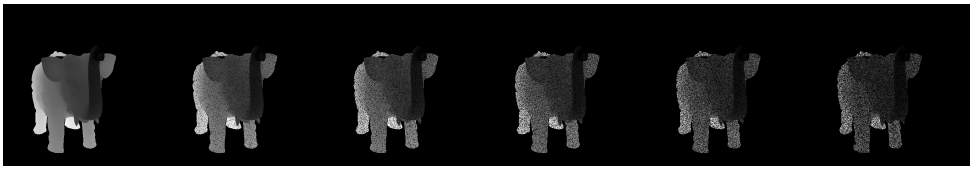


FIGURE 4.6: Noise-intensity on $\mu-0, \mu-10, \mu-20, \mu-30, \mu-40, \mu-50$,. Object Name: elephant-zun-lid.

4.3.4 Fit to PyTorch

In order to saving the training time, the dataset is compressed 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.

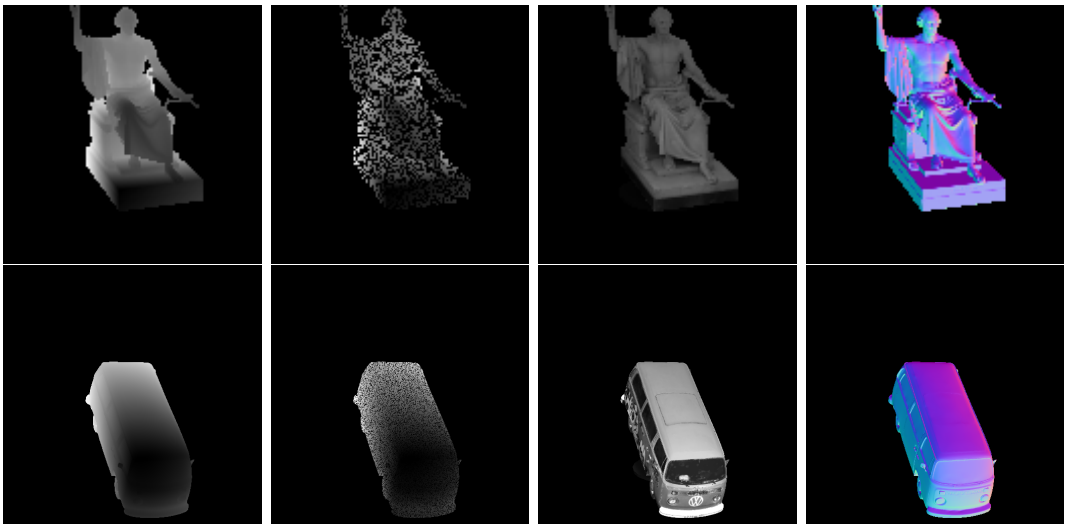
Name	**Content**
input-tensor	Vertex Image Light Direction
output-tensor	GT-Normal Image GT-Light-Direction
Light position	light position
Camera Matrix	K,R,t
Depth Range	minDepth, maxDepth

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 $128 \times 128 \times 1$. The depth map is converted to 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 has corresponding 3000 training cases. Each scene has a corresponding ground-truth normal map for loss calculation and the evaluation.



(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

The training processes are evaluated in every epochs with 100 evaluation scenes that models never seen before, which contains the 5 different objects in the "synthetic-50-5" test set. Figure 5.1 shows some of the test scenes during the training work. 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.

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 (Kingma and Ba, 2014), learning rate start from 1×10^{-3}, learning schedule [8,1000], 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 evaluation on the test dataset converged.

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 based on geometry information. 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 verify the performance the gated convolution layers. 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, since 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. Since it is more complicate than GCNN, 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 $1e-3$. 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.

We also 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 weaker than the other two models.

However, the sacrifice of accuracy gives a relatively lighter model. Since we remove the fusions between different pipes, the corresponding upsampling layers in the image and light pipes can also be removed. The model can be trained faster 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

We evaluated our models on synthetic dataset with resolution 128×128. Based on metrics proposed by Fouhey, Gupta, and Hebert, 2013, 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 robust 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.

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 evaluate the 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 metric 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 model.

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 Visualization 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 **??**. 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 neighbor based method is depended on a well-chosen parameter k.

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

Figure **??** 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 $k = 2$.

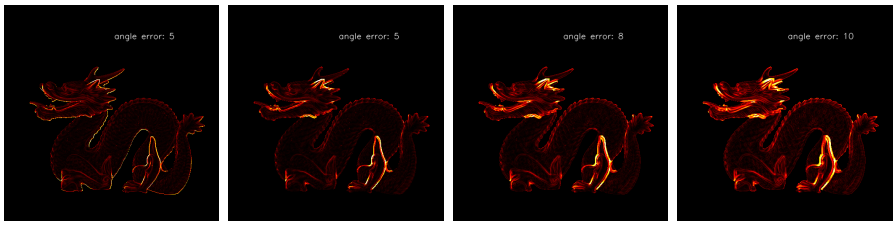


FIGURE 5.3: Error map of neighbor based method with different k values. From left to right, $k = 1, 2, 3, 4$ separately.

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.

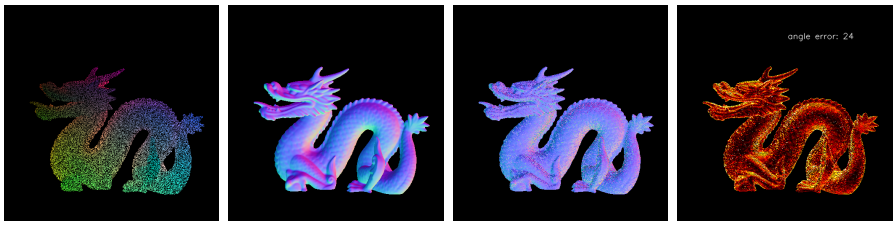
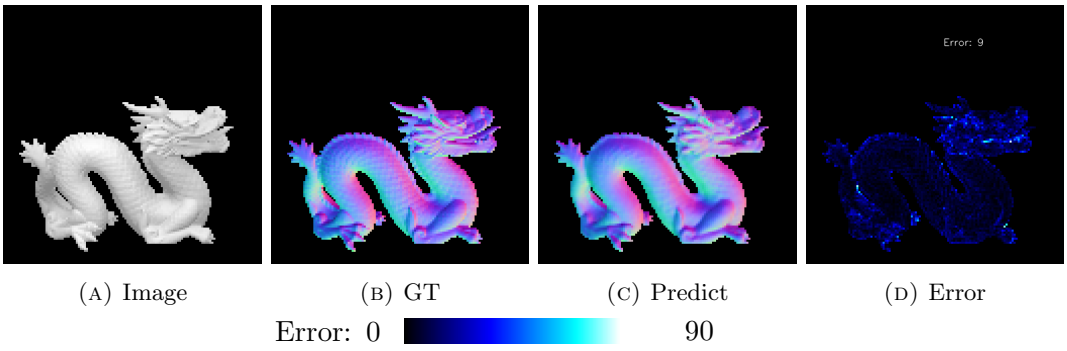


FIGURE 5.4: Evaluation of neighbor based method on a noised dragon model

5.4 Visualization evaluation on GCNN model

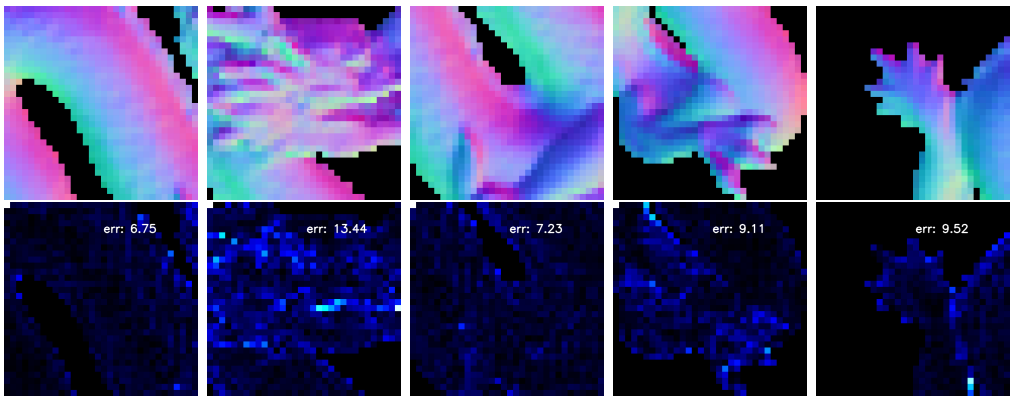
A qualitative evaluation on object "dragon" is shown in Figure 5.5. As shown in the figure, GCNN model achieved a mean angle error in 9 degrees on this dragon object. The image has an overall good performance on the whole object. A closer evaluation is shown in Figure 5.6, the normal accuracy especially 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, the head of dragon object, CNN model gives a much brighter error map (which lead to a higher angle error).

Figure 5.12 shows more evaluation on GCNN model. Although 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.



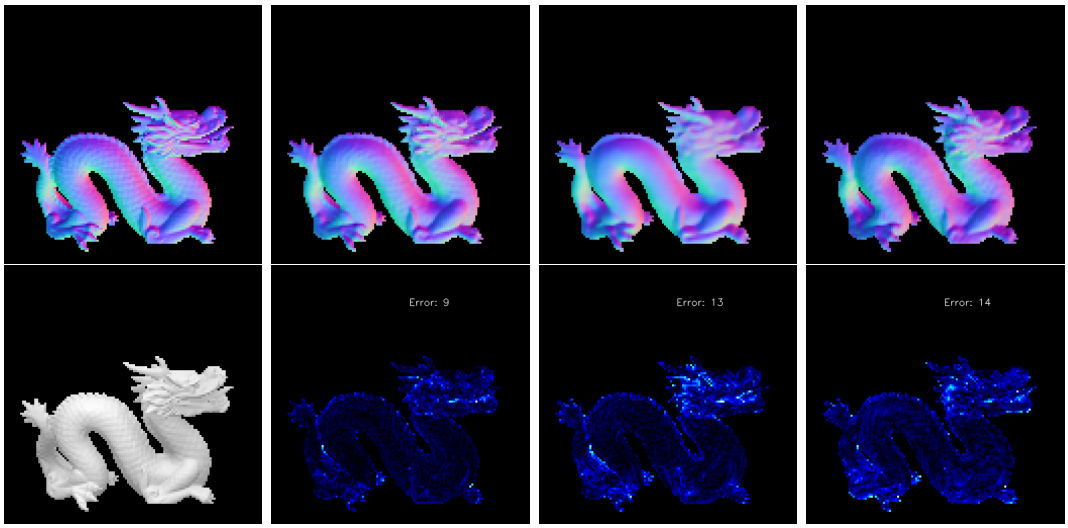
(A) Image (B) GT (C) Predict (D) Error

FIGURE 5.5: Normal inference based on GCNN. Test image has resolution 128×128



(A) Scale (B) Head (C) Foreleg (D) Hindleg (E) Tail

FIGURE 5.6: Zoom in of some regions of Dragon object. The first row is surface normal, the second row is the corresponding errors. Zoom-in normal map corresponding 32×32 points in the original matrix.



(A) GT (B) GCNN (C) NOC (D) CNN

FIGURE 5.7: Comparison of GCNN model with no skip connection version(NOC) and standard convolution layer only version (CNN). The second row is the corresponding mean average degree error.

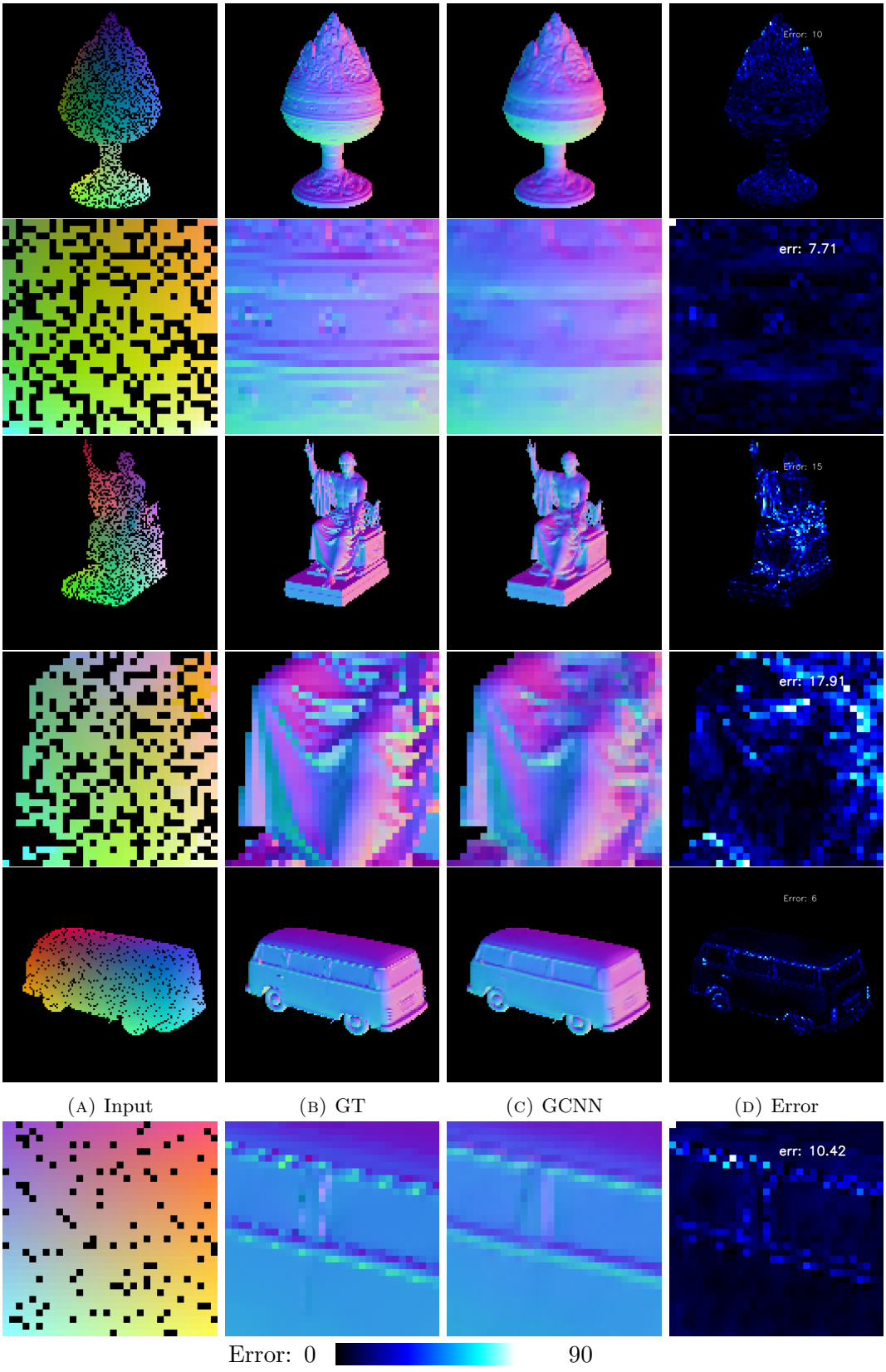
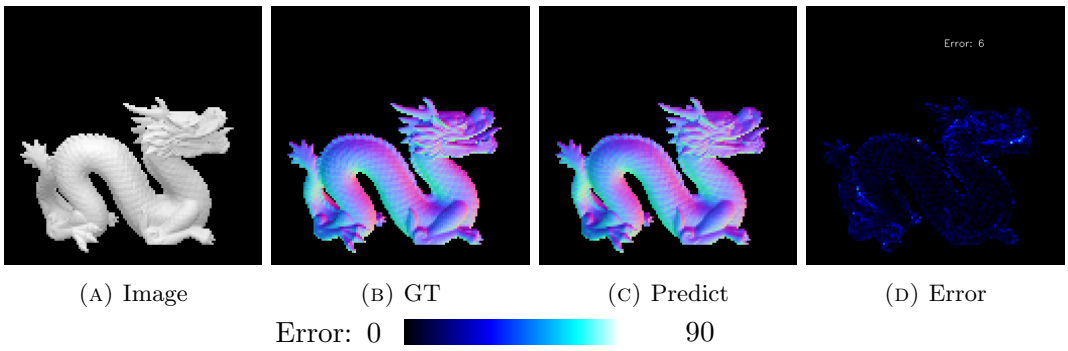


FIGURE 5.8: Evaluation on objects Baoshanlu, Washington statue, Bus(from top to bottom).

5.5 Visualized evaluation on Trip-Net models

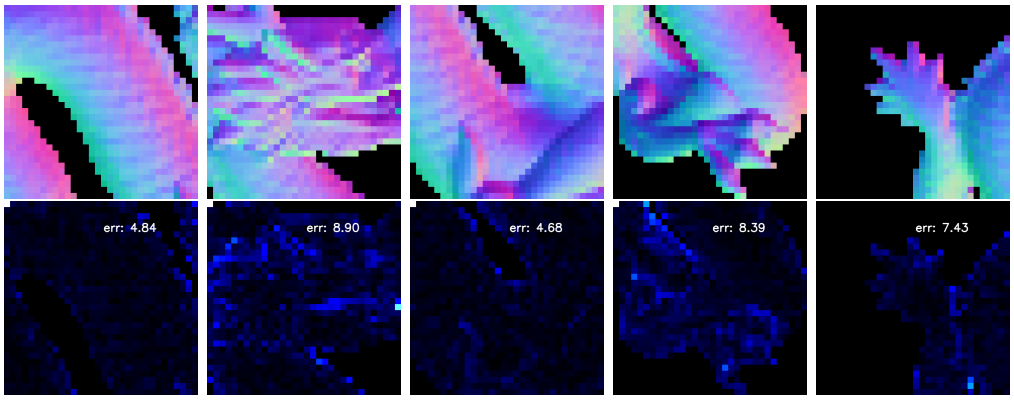
For the approach using illuminated calibrated RGBD image, the task is undertaken by Trip-Net introduced in **??**. The Trip-Net uses illuminated information align with the geometry information achieves a sharper and more accurate result compare to the GCNN model. The qualitative evaluation is shown in 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 degree whereas the GCNN is 9.

As shown in the figure 5.11, 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. It seems that the illumination helps the model to learn sharper features in the network.



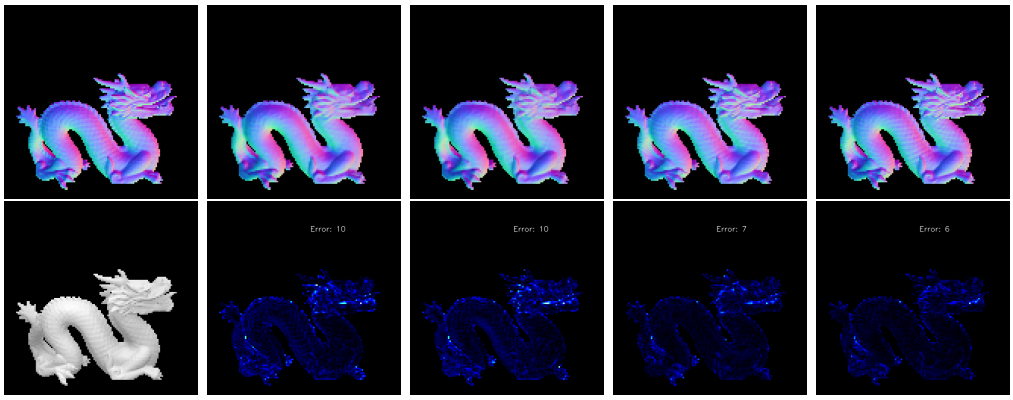
(A) Image (B) GT (C) Predict (D) Error

FIGURE 5.9: Normal inference based on Trip-Net. Test image has resolution 128×128



(A) Scale (B) Head (C) Foreleg (D) Hindleg (E) Tail

FIGURE 5.10: Zoom in of some regions of Dragon object. The first row is surface normal, the second row is the corresponding errors. Zoom-in normal map corresponding 32×32 points in the original matrix.



(A) GT (B) F1 (C) F2 (D) F3 (E) F4

FIGURE 5.11: Comparison between different fusion times for Trip-Net. The first row is surface normal, the second row is the corresponding errors. The number in "Fx" represents the fusion times.

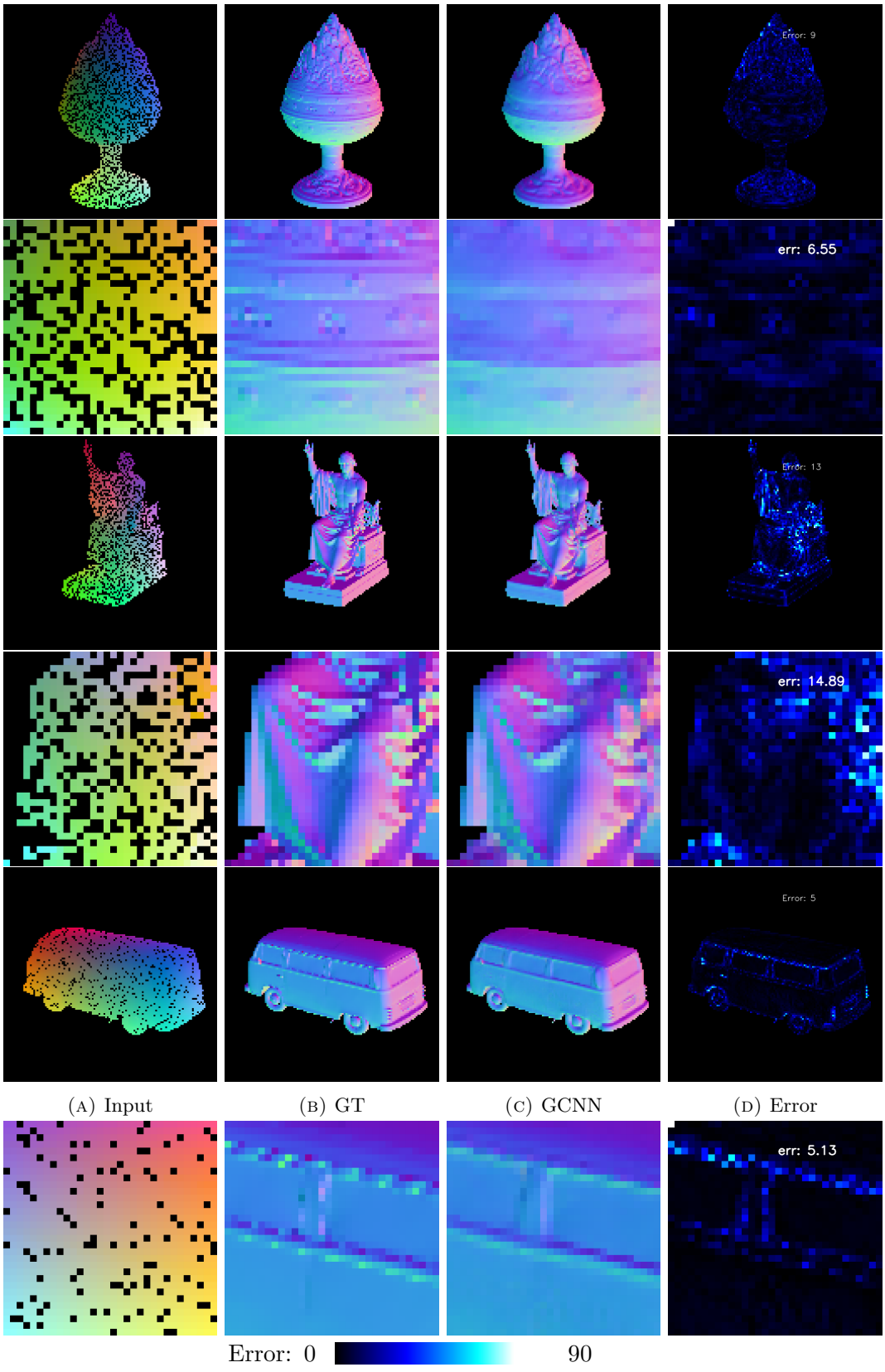
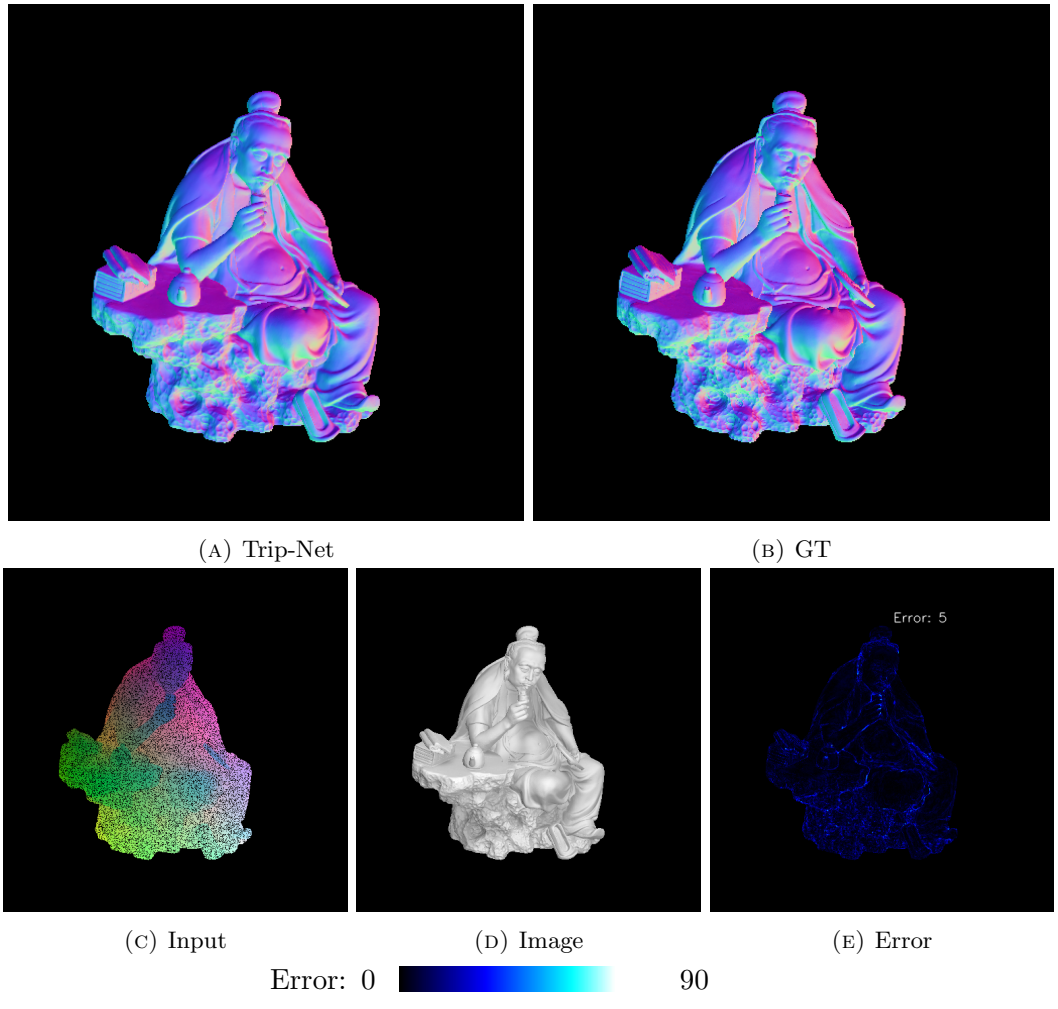


FIGURE 5.12: Evaluation on objects Baoshanlu, Washington statue, Bus(from top to bottom).

5.6 Trining on higher resolution

We also prepared the dataset with resolution 512×512 for model evaluation. Higher resolution 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 will more smooth and easy to calculate 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. Thus the higher resolution helps the model to calculate a more accurate normal map.



(A) Trip-Net (B) GT

(C) Input (D) Image (E) Error

FIGURE 5.13: Normal inference based on Trip-Net. Test image has resolution 128×128

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

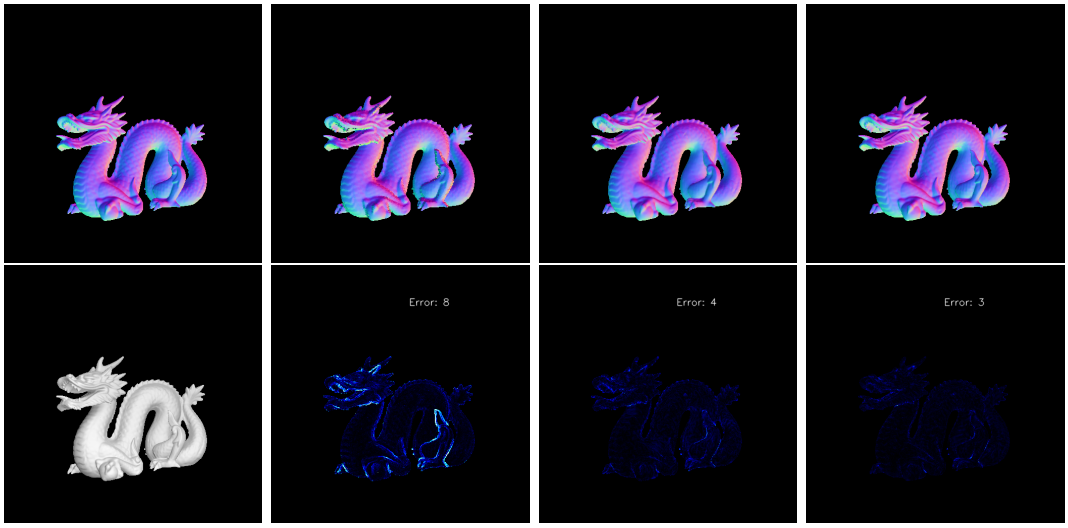
Metrics	SVD	GCNN	Trip-Net
Mean	8.88	5.82	5.33
Median	3.66	3.98	3.67
5°	0.63	0.63	0.66
11.5°	0.79	0.89	0.91
22.5°	0.89	0.97	0.97
30°	0.92	0.98	0.99

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

training work on dataset with 512×512 resolution. The training on high resolution network takes longer time but achieves a lower angle error.

A quantitative evaluation is shown in table 5.10. It follows the same performance rank compare 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 for Trip-Net and 5° mean error. A qualitative evaluation is shown in figure 5.13, this is a figure object with both smooth area (the belly and arm) and highly detailed area with sharp surface (the uneven surface of stone table). Our models gives an average degree area at 5°. A comparison with other models is shown in figure 5.14.

Another remarkable result is the SVD approach, which also gives a good result (8.88° in mean degree metric). Like we discussed, the surface are relative 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 are mainly on a relatively flat plane thus they are good enough for an accurate normal inference. Consequently, we can see from the error map in Figure **??**, the sharp edges of the dragon, like the horns and the hindlegs, the SVD approach still has high errors.

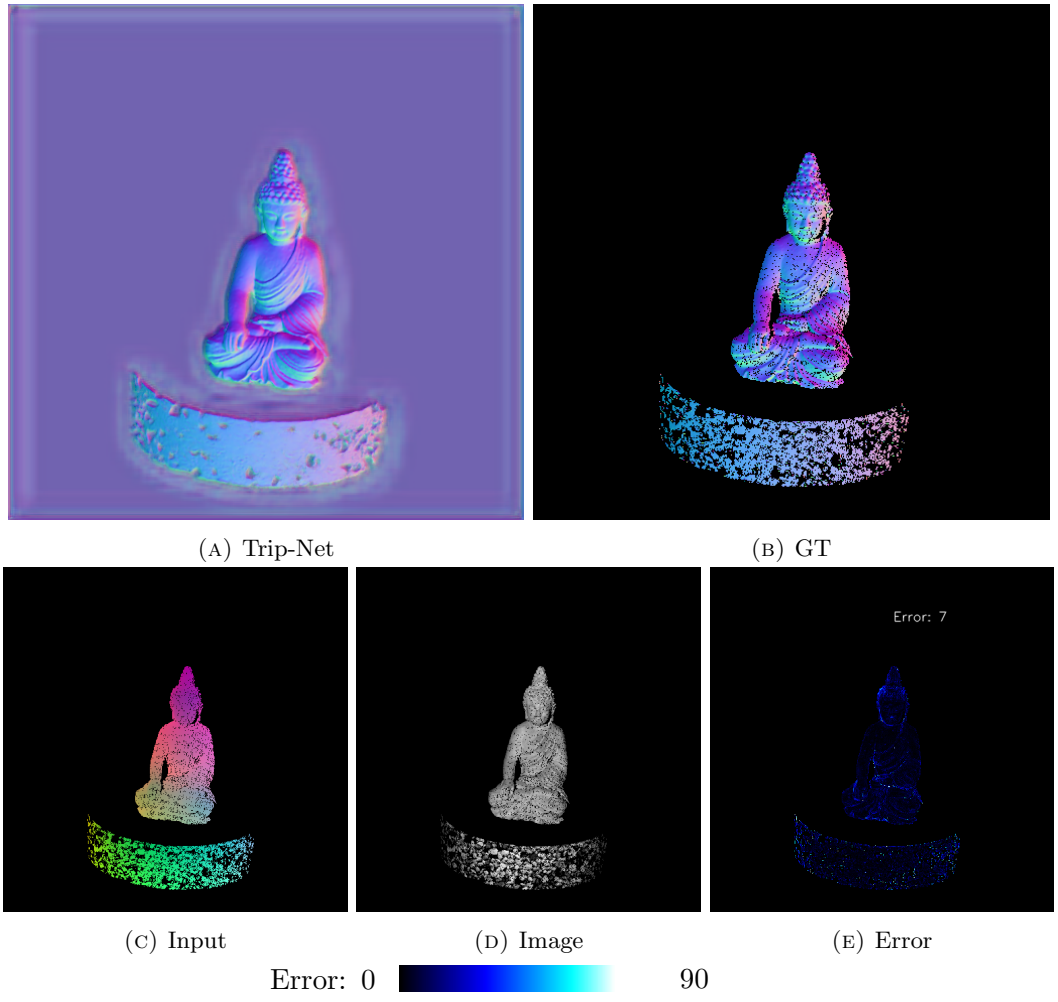


(A) GT (B) SVD (C) GCNN (D) Trip-Net

FIGURE 5.14: Comparison between different SVD, GCNN and Trip-Net model on 512×512 dataset. The first row is surface normal, the second row is the corresponding errors.

5.7 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 the applicability of our approaches. 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.



(A) Trip-Net (B) GT

(C) Input (D) Image (E) Error

FIGURE 5.15: Normal inference based on Trip-Net. Test image has resolution 512×512

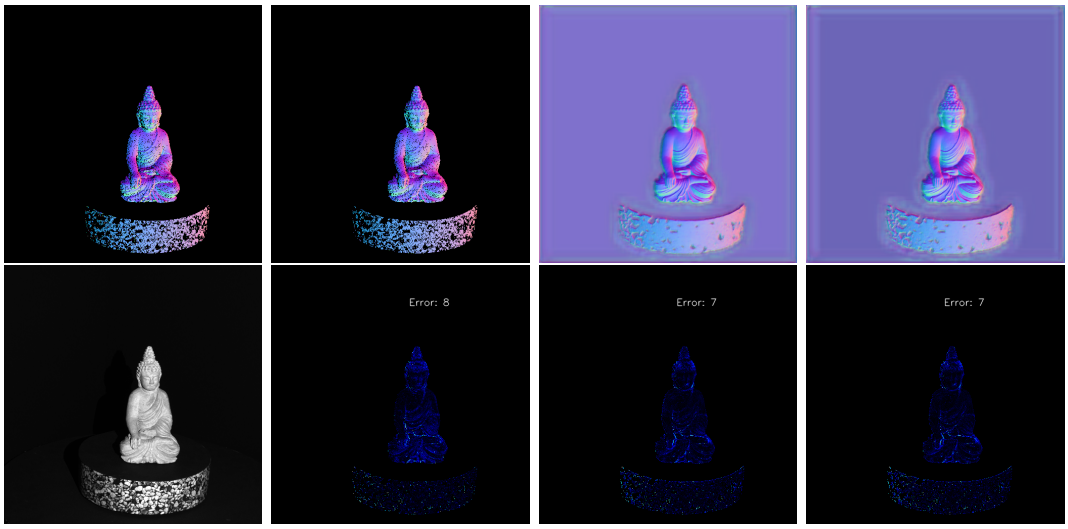
Since in the real-dataset, we don't have a ground-truth for evaluation. But if we see directly from the visualization in figure 5.15, we can see that Trip-Net approach has good ability to "mend" the missing pixels in the scenes and also gives a sharp result. The folds on the gowns are recognizable and even countable. 5.16 compares the Trip-Net with other approaches.

But we also noted that the big holes in the base stage remains in the depth maps. These big holes corresponds to the shinny and dark texture area, which common exists in the depth map captured by the scanners. Since these big holes are related

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.

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 random binary mask. Thus our models failed to mend missing big holes in the estimated normal map. This can be a further direction of this topic, that find a way to generate high similar depth map noise to get a more robust training dataset.



(A) GT (B) SVD (C) GCNN (D) Trip-Net

FIGURE 5.16: Comparison between different SVD, GCNN and Trip-Net model on 512×512 real dataset. The first row is surface normal, the second row is the corresponding errors.

Chapter 6

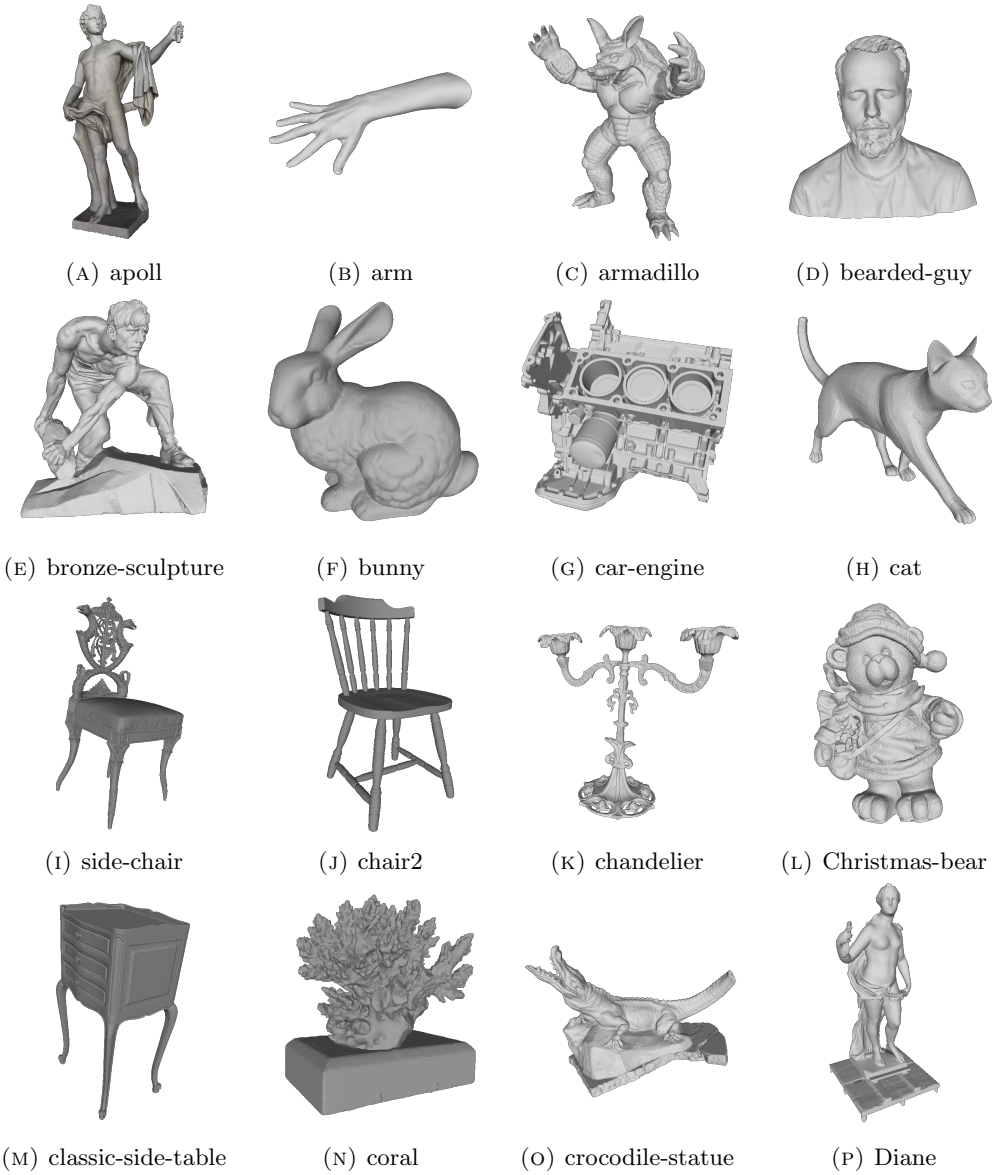
Conclusion

Gated convolution neural network...

Appendix A

Dataset

A.1 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 training dataset A

(A) Dresdner-knabe

(B) fantasy-dragon

(C) fox-skull

(D) Garuda and Vishnu

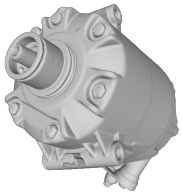
(E) giraffe-skull

(F) green-hat

(G) happy-budda

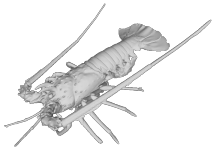
(H) heart-pendant

(I) indonesian-statue

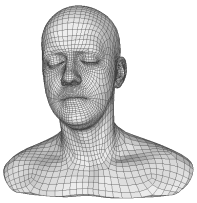


(J) industrial-compressor

(K) kylix



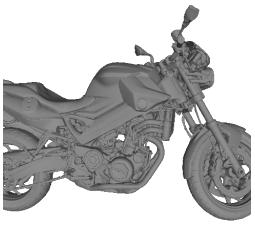
(L) lobster



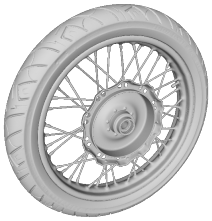
(M) head

(N) Lu-Yu

(O) Maitreya

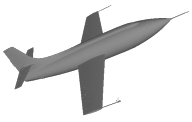


(P) motorbike



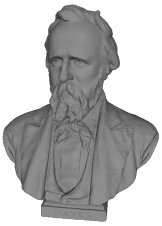
(Q) motorcycle-wheel

(R) Pan



(S) plane

(T) elkins-refrigerator



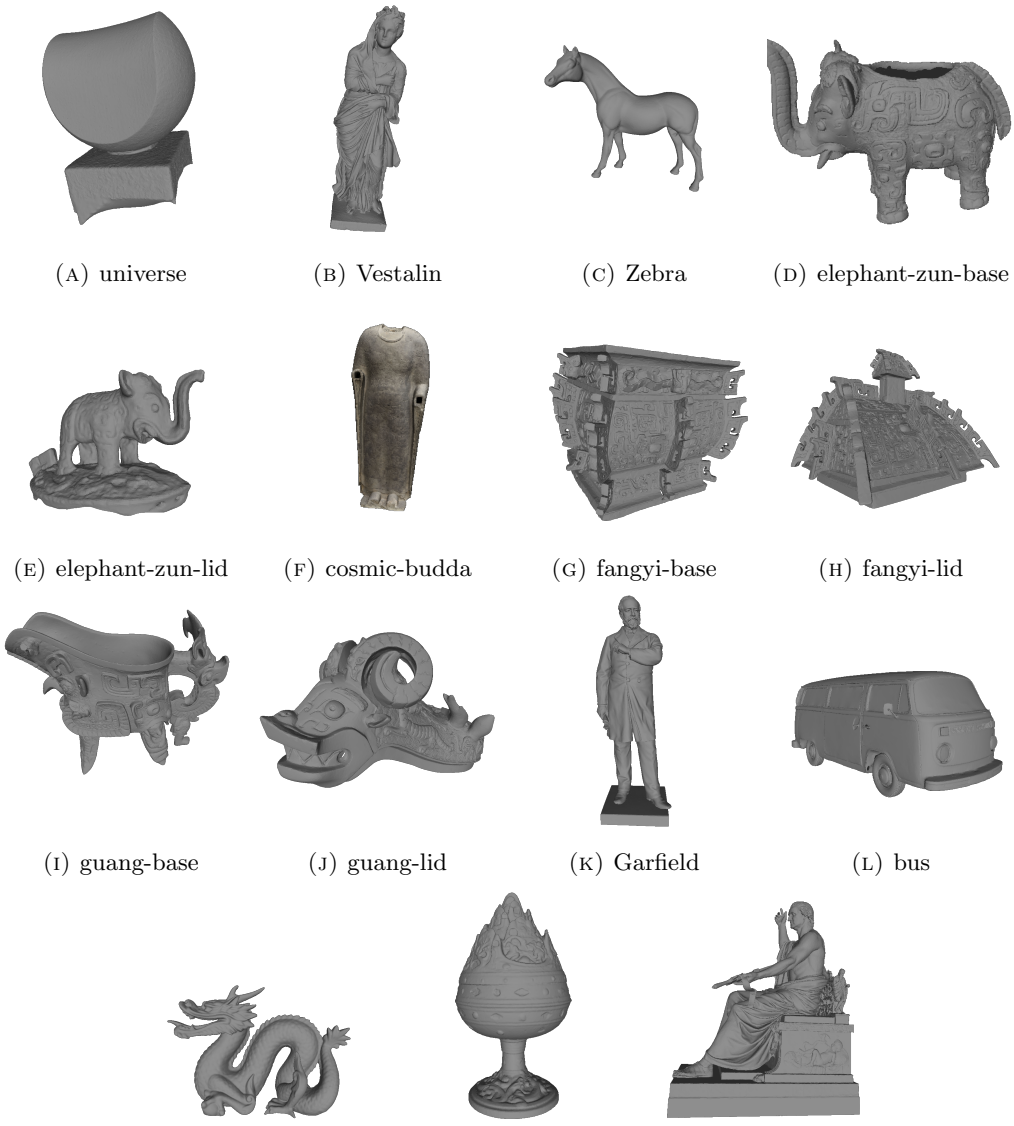
(U) Rutherford

(V) statue-dragonfly-tamer

(W) Tiger

(X) transmission

FIGURE A.2: Point clouds in training dataset B



(A) universe (B) Vestalin (C) Zebra (D) elephant-zun-base

(E) elephant-zun-lid (F) cosmic-budda (G) fangyi-base (H) fangyi-lid

(I) guang-base (J) guang-lid (K) Garfield (L) bus

(M) dragon (N) baoshanlu (O) Washington

FIGURE A.3: Point clouds in training dataset C

Bibliography

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

Ben-Shabat, Yizhak, Michael Lindenbaum, and Anath Fischer (2019). "Nesti-Net: Normal Estimation for Unstructured 3D Point Clouds Using Convolutional Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

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

— (2020b). "Confidence Propagation through CNNs for Guided Sparse Depth Regression". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.10, pp. 2423–2436. DOI: 10.1109/tpami.2019.2929170. URL: https://doi.org/10.1109%2Ftpami.2019.2929170.

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

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

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

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

Holzer, S. et al. (2012). "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*, pp. 2684–2689. DOI: 10.1109/IROS.2012.6385999.

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

Hua, Jiashen and Xiaojin Gong (2018). "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, 2283–2290. ISBN: 9780999241127.

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

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

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

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

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

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

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

McGuire, Morgan (n.d.). *Artec3D*. URL: https://www.artec3d.com/3d-models/art-and-design.

— (2017). *Computer Graphics Archive*. URL: https://casual-effects.com/data.

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

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

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

Redmon, Joseph and Ali Farhadi (2018). *YOLOv3: An Incremental Improvement*. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767.

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

Silberman, Nathan et al. (2012). "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, pp. 746–760. ISBN: 978-3-642-33714-7. URL: https://www.microsoft.com/en-us/research/publication/indoor-segmentation-support-inference-rgbd-images/.

Smithsonian 3D Digitization (n.d.). URL: https://www.3d.si.edu/explore.

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

The Stanford 3D Scanning Repository (n.d.). URL: http://www.graphics.stanford.edu/data/3Dscanrep/.

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

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

Yang, Na-Eun, Yong-Gon Kim, and Rae-Hong Park (2012). "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)*, pp. 658–661. DOI: 10.1109/ICSPCC.2012.6335696.

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

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

Zhou, Jun et al. (2021). *Fast and Accurate Normal Estimation for Point Cloud via Patch Stitching*. arXiv: 2103.16066 [cs.CV].