Technische Universität Kaiserslautern

Master Thesis

Improved Normal Inference from Calibrated Illuminated RGBD Images

Author:
Jingyuan Sha

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

Augmented Vision
Department of Computer Science

September 14, 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:

Abstract

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

This approach extends the geometry-based approach with photometric stereo analysis techniques using calibrated illuminated RGB-D images for normality estimation. Based on photometric and geometric information as input, our model can improve the accuracy of the geometry-based approach on our dataset by 10% (mean angular error). One of the most important points of our approach is that the proposed method requires only one light source for each scene detection instead of multiple light sources, which is very easy to configure in practice.

Our model is trained on a neural network with gated convolution (trip-net). The gated convolution design is intended to handle the missing pixels in the depth map, which are a common noise of the depth sensor. The network has a full convolutional architecture, so it can handle data with different resolutions as input. The output of our model is the corresponding normal map with a 1:1 correspondence to the input data. To train our models, we also created a dataset of 55 highly detailed 3D objects called *Synthetic-50-5*. We trained our model on this synthetic dataset and then applied it to a real dataset. As our qualitative evaluation shows, the performance of the presented method can be observed especially in detailed regions. Finally, a quantitative evaluation with 6 metrics is performed in this thesis.

Contents

Chapter 1

Introduction

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

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

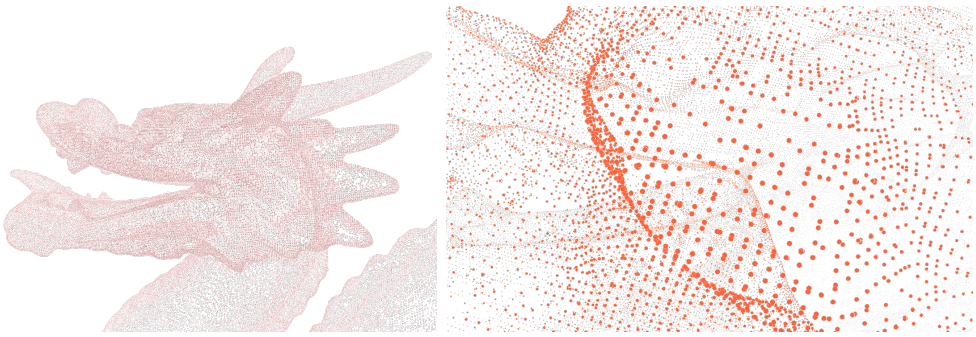


Figure 1.1: Left: A portion of the point cloud of the *dragon* object. Right: Zoom in of the point cloud.

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

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

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

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

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

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

Chapter 2

Related Work

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

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

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

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

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

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

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

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

Chapter 3

Approaches

The normal inference task involves estimating the surface normal of a 3D object, assuming that the surface of the object is mathematically unknown. This means that we cannot simply use the cross product of two non-parallel surface vectors to find the normal. Instead, the surface is sensed by digital sensors under certain types of data, which can be divided into two groups. The first group is geometry data, where a point cloud is used to capture the geometry information of the surface in 3D space. The approach that uses point clouds for normal estimation is called geometry-based approach. Another type of data is images that record the reflectance and shading of the surface under the following lighting conditions. The surface normal is calculated based on specific light models such as BRDF. The other approach, which uses the relationship between reflectance and shading to calculate the surface normal, is called photometric stereo.

In this chapter, the geometric and photometric stereo based approaches are presented separately. Then, a method for estimating the surface normal using both types of data based on deep learning approaches is introduced.

3.1 Notations

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

3.2 Geometry-based Normal Estimation

Geometry-based normal estimation uses the point cloud of the object surface as input. The task can be formulated as follows.

Given a structured point cloud $\mathbf{V}^{W\times H\times3}$, in which the 3D point positions are stored with respect to the image grid. We want to compute the corresponding normal map $\mathbf{N}^{W\times H\times3}$. The normal map is simply a matrix representation of the normals of all points in the point cloud. A normal $\mathbf{n}^{3\times1}$ at the point $\mathbf{p}^{3\times1} \in \mathbf{V}$ is a unit vector whose direction is outward from the surface and perpendicular to the tangent plane of the surface at point $\mathbf{p}$.

The idea behind the neighborhood-based method is to fit a plane $\tilde{\Pi}$ containing the k neighbors $\mathbf{p}_1, ..., \mathbf{p}_k \in \mathbb{R}^{3\times1}$ of the point $\mathbf{p}$, then compute the normal $\tilde{\mathbf{n}}$ of the plane $\tilde{\Pi}$. The $\tilde{\mathbf{n}}$ is considered as an approximation to the normal $\mathbf{n}$ of the tangent plane Π.

The point and its neighbors are assumed to lie in the same plane Π, which is exactly the tangent plane of the surface at the point $\mathbf{p}$. This is usually not true for most of the surface, since the surface is usually not flat, but has some degree of curvature. But if k is of a suitable scale and the point cloud is dense enough, we can approximately consider all k neighboring points in a small area 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 above assumption, the task of normal inference is trivial, since the surface is then mathematically describable. We can generate the approximate tangent plane at each point with its neighbors, then the normal $\tilde{\mathbf{n}}$ of the tangent plane can be derived from the following equations.

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

where

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

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

If we consider more than 3 neighbors, the points are usually not on the same plane. In this case, the system of equations is overdetermined. We can use the optimization approach to find a plane that has the least distance to all points in 3D space. By $\mathbf{A} \in \mathbb{R}^{k \times 3}$ we denote the vector matrix vertically stacked by the k vectors we found on the approximated tangent surface. Then the system of equations is transformed into an optimization problem

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

where the additional condition is added to avoid trivial solutions, and the normal is supposed to be a unit vector.

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

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\boldsymbol{T}}$$

The last column of $\mathbf{V}$ is the solution of the system of equations and consequently the surface normal.

The optimization based approach using SVD is more robust compared to selecting only 3 points for normal computation, since the plane generated by 3 neighbors is not guaranteed to be a good approximation to the actual tangent plane. However, the size of the neighbors k is a key parameter of this approach. For the smooth surface, a relatively larger k can help the system of equations find a plane that is closer to the tangent plane. For the sharp surface, k must be small enough to ensure that the points it contains are still approximately in the same plane, otherwise the more points it contains will just be the outliers for the equation solution.

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

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

Since the above approach is essentially solved by SVD, this method will be referred to as *SVD* in the remainder of the paper for simplicity.

3.3 Photometric Stereo based Normal Estimation

Photometric stereo was originally introduced by [33] and is a completely different approach from the geometry information based approach such as *SVD*. It estimates the surface normal of the object by observing the object in the same position under different illuminated conditions. It is based on the fact that the light reflected from a surface depends on the surface normal and the light direction.

Before presenting this approach, let us discuss what we can actually obtain from the images. An image $\mathbf{I}$ can be decomposed into two parts: the reflectance $\mathbf{R}$ and the shading $\mathbf{S}$,

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

where $\odot$ denotes the element-wise product. This decomposition of the image is based on the intrinsic image model proposed by [1]. It interprets the observed image into a reflection image and a shading image. Figure 3.1 shows a visualization of the decomposition.

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

The equation can be further decomposed based on different surface models. If we assume that the object surfaces are Lambertian surfaces, i.e. surfaces that reflect light in all directions, as shown in Figure 3.2, then the reflectance is equal to the albedo ρ, the shading image can be decomposed as the product of the radiance of the incident light L_0 and the cosine of the angle of incidence, which is the dot product of the surface normal $\mathbf{N}$ and the light source direction $\mathbf{L}$.

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

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

The equation can be rearranged as follows

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

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

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

If we use a set of k images for the same scene, taken based on different light projections. Then, for each pixel (x, y) in a scene, we can set up a system of equations

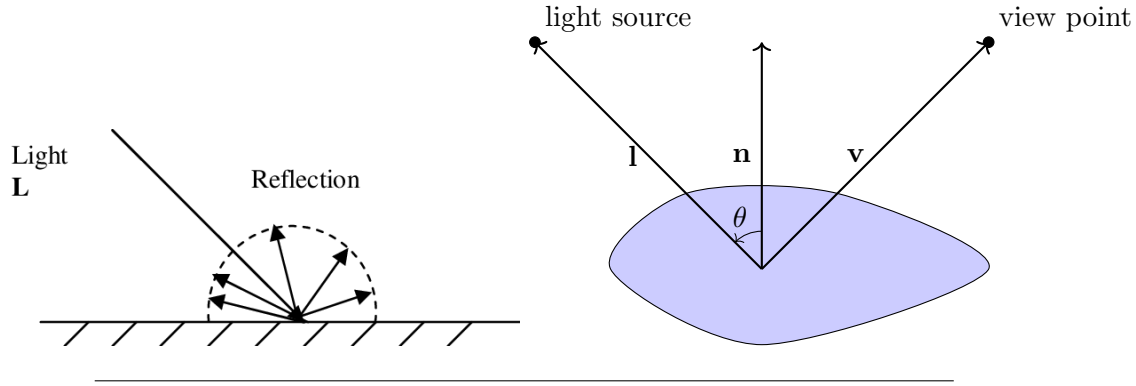


Figure 3.2: Left: the reflection of light on a Lambertian surface. (Image source [20]). Right: the surface normal, the light direction of the source and the viewing direction, where θ denotes the angle between the light direction and the normal.

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

For simplicity, $\mathbf{l}_i^T$ for $1 \leq i \leq k$ denotes the direction of light at position (x, y) in k-th image . The equation can be solved using the least squares method. Then, the albedo can be determined with the light intensity, since the normal is a unit vector,

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

Then the normal can be determined as follows

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

We compute the surface normal for each point to obtain the surface normal map.

This approach is also called Shape from Shading (*SFS*) [11]. Both normal maps and the corresponding albedo are obtained from a series of images under different lighting conditions. Since the direction of light is used in the calculation, the cameras and the position of the light source must be calibrated beforehand.

In the shape from shading approach, at least three light positions are considered for normal estimation on a pixel. It is also common to use more than three light sources to cover most points of the surface as much as possible.

We assume that in each neighborhood there is a coherent behavior for the albedo that can be modeled by the filters in the neural network, and in the following section we propose an learning based approach for normal inference from geometry information, but using only one illuminated scene.

3.4 Gated Convolution Neural Network for Surface Normal Estimation

Recently, Deep Learning based methods have achieved great success in image processing[26] [30]. These network architectures use a stack of RGB/grayscale images as input and are used for classification problems. Typically, the images are convolved with convolutional layers and sampled with pooling layers. The output values of the networks consist of a single value representing the index of the corresponding class, or a set of values representing the position of the bounding boxes[26]. However, in many other image processing tasks, such as inference of normal maps, the output is required to have the same form as the input, while each pixel in the output image requires a prediction. Therefore, the output consists not only of one or several labels, but of a matrix similar in size to the input. In this case, the traditional network architecture with full connections in the last layers is no longer suitable for label prediction.

[27] has proposed an architecture called *U-Net* for biomedical image segmentation. The architecture is shown in Figure 3.3. This net has a very regular architecture for data processing, while the down/up sampling part has a similar design. For the feature extraction part, called down-sampling, the architecture follows the traditional CNN architecture. For the up-sampling part, the network uses an architecture similar to the down-sampling operations, which uses the same convolutional operations but replaces the max-pool layers with up-conv layers. An important design is the skip connection in the up-sampling part. The feature maps extracted in the down-sampling part are concatenated with the feature maps in the up-sampling part. This helps the network to find the locations in the original image and use them to predict a sharp output.

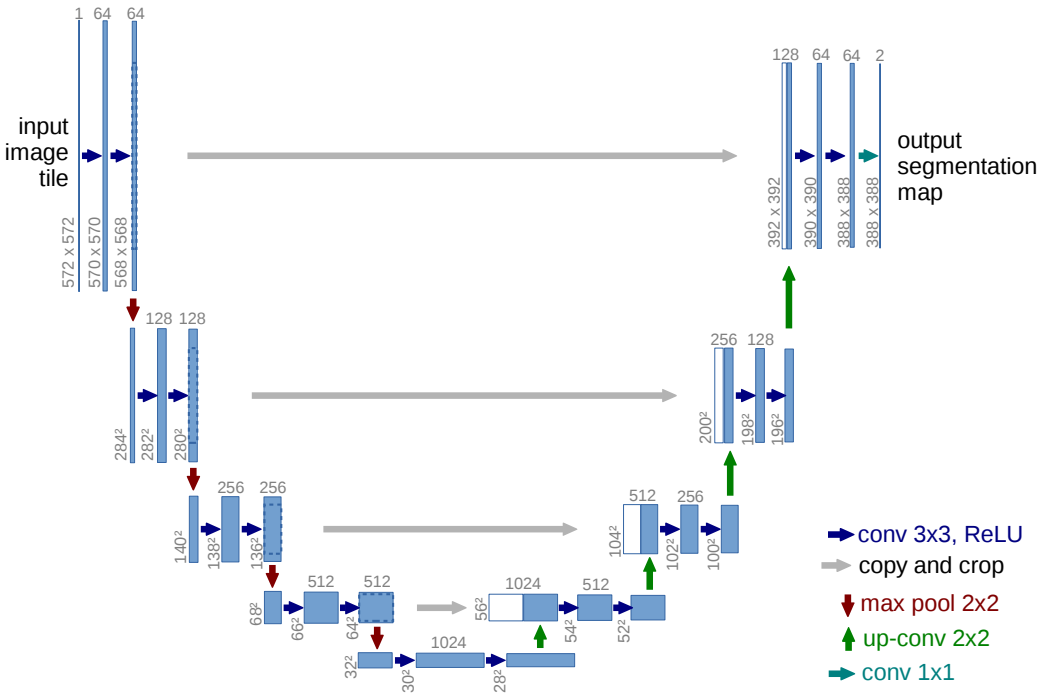


Figure 3.3: The structure of UNet. (Image source:[27])

The *U-Net* relies on standard convolutional layers to construct the network. This is useful for image processing tasks with fully dense input data, since there are no

missing pixels. However, for some other input data, such as depth maps acquired by light scanners, it is not always fully dense data, but data with many missing pixels and holes. In this case, we can still apply the standard convolutional layers to extract the feature maps from the semi-dense depth map, but then the valid pixels and the invalid pixels (the missing pixels) will be mixed up during training, resulting in blur and color discrepancy as mentioned by [19]. Therefore, it is useful to find a way to distinguish the valid and invalid pixels during training.

[5] use a binary mask to denote valid pixels, and also use normalized convolution as a surrogate for the default convolutional layer in the training model. The normalized convolution is represented 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 the input, (i, j) is the displacement in the kennel, and M is the corresponding mask. A binary mask uses 1 to indicate valid pixels, and 0 otherwise. $\odot$ denotes element-wise multiplication.

3.4.1 Gated Convolution

[23] proposed a gated activation unit to model more complex interactions compared to standard CNN layers. This was mainly inspired by the multiplicative units that exist in the Long Short-Term Memory ($LSTM$) proposed by [9] and the Rated Recurrent Unit (GRU) proposed by [2]. [35] has deployed the same gated unit and uses it as a gated convolution layer for training tasks with incomplete input data, such as image inpainting tasks. In this gated convolution layer, the mask used to distinguish between valid and invalid pixels is not specified in advance, but learned during training. The advantage is that the erroneous measurements that are not masked in the depth data can be learned during training to further improve performance.

The structure is shown in Figure 3.4. Instead of using a mask as input to indicate valid pixels, a standard convolutional layer is used to learn this mask directly from the data, and a sigmoid function is used as an activation function to indicate the confidence of pixel validation. Meanwhile, another standard convolutional layer is set aside to learn the feature maps with a ReLU/LeakyReLU activation function. Therefore, both the mask and the input features are learned during training. Then, an element-wise multiplication is performed 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 the input (N, C_{in}, H, W) and the output $(N, C_{out}, H_{out}, W_{out})$:

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

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

corresponding to the k-th input channel, $b(C_{o_j})$ denotes the bias of the j-th output channel.

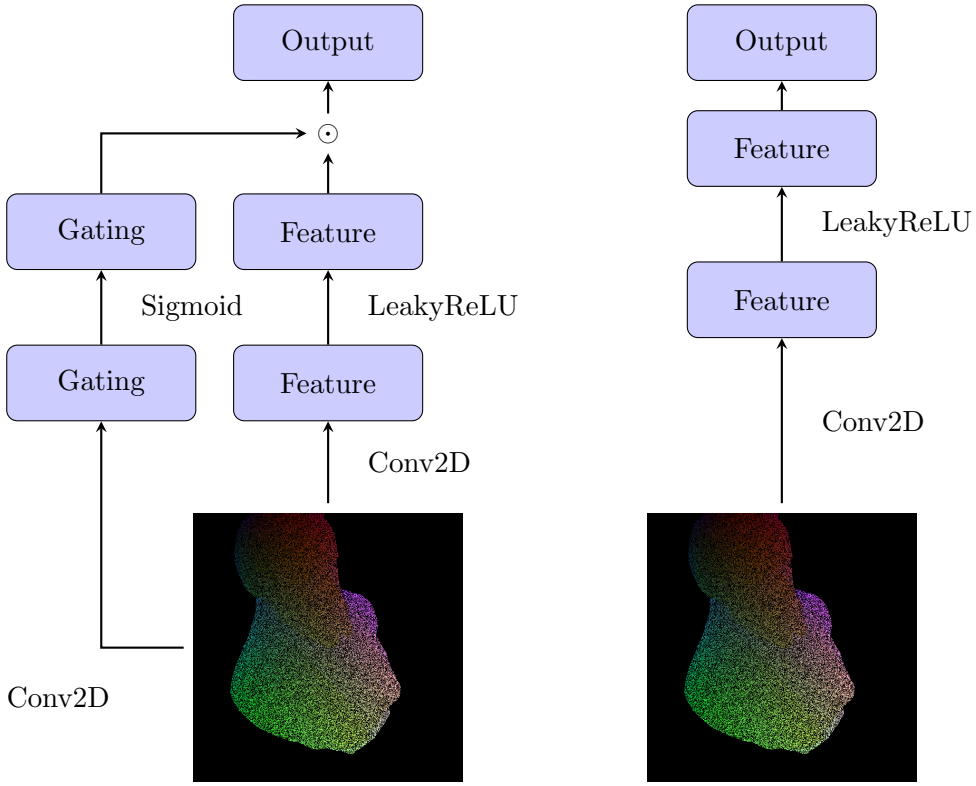


Figure 3.4: Left: Gated Convolution Layer, where ⊙ denotes element-wise multiplication. Right: standard convolution layer.

3.4.2 GCNN Architecture

Based on the above implementation, we proposed a network based on the *U-Net* proposed by [27], replacing the standard convolutional layers with gated convolutional layers used for semi-dense normal inference tasks, called gated convolution neural network (*GCNN*), as shown in Figure 3.5.

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

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

After feature extraction, the network upsamples the feature map three times to obtain the output matrix $\mathbf{X}_{out}$. Each upsampling consists of an interpolation operation that uses nearest neighbor interpolation for upsampling the feature map, and a concatenation layer that concatenates the interpolated result and the corresponding

high-resolution feature map $\mathbf{X}_{df_1}, \mathbf{X}_{df_2}, \mathbf{X}_{df_3}$ from the downsampling part. This is also called a skip connection. In the last step, a gated convolution layer is used to reduce the channel size to fit the next upsampling block. After upsampling three times, the resolution goes back to the original size. After that, two standard convolutional layers without activation function are added to predict the surface normals. The entire upsampling branch can be represented as a regression function n,

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

All convolutional layers of the network use the same kernel size 3×3.

One of the main features of the network is that the output has the same size as the input. This is achieved by (1,1) padding and the same number of channels in the convolution layers. Thus, the surface normal map can be estimated with the same dimension as input data. Another important point of the network is its robustness to noise using gated convolutional layers. The network can take a semi-dense matrix as input and then predict the fully dense matrix as output. The last feature is the multipurpose use of scenarios. No specific input type is given in the description. The network is also fully convolutional and therefore 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 process the data. In our work, the input refers to a structured point cloud. Therefore, it is suitable for a geometry-based approach that uses the point cloud as input and estimates the corresponding surface normal. However, as mentioned in the introduction, the goal of this work is to discuss the improvement of normal inference based on an illuminated calibrated RGB-D image, i.e., we want to see if we can improve the performance of normal estimation not only based on geometry information such as the point cloud or the depth map, but also with illumination information such as the image and the light map. Therefore, we need to find an architecture that takes into account all these types of data.

3.5.1 Light Map, Gray-scale Image and Vertex Map

First, we present the required input types for our network.

Vertex Map The vertex map $\mathbf{V}$ is converted from the depth map presented in chapter 4, while the depth map is acquired from a depth camera. It contains a set of surface point locations in 3D space. Each pixel in the vertex map corresponds to a point position. These vertices contain the geometry information of the object surface, which can be further used for training deep learning models. For each scene, the camera takes only one image, leaving only one vertex map. However, as mentioned earlier, the vertex map is only semi-dense. We use such a semi-dense vertex map as input for the surface normal estimation.

Grayscale Image The grayscale image $\mathbf{I}$ is acquired in correspondence with the depth map using the same calibrated camera equipment, so the intrinsic matrix $\mathbf{K}$ and the extrinsic camera matrix $[\mathbf{R}|\mathbf{t}]$ are known. Unlike the vertex map, it is usually completely dense.

Light Map The light map $\mathbf{L}$ can be derived from the vertex map $\mathbf{V}$. The position of the light source (s_x, s_y, s_z) is given by the input data. As shown in Figure 3.2, the direction of the incident light is a vector point from the light source to the surface point and can therefore 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 $\mathbf{V}$ refer to the camera space. The light direction map $\mathbf{L}$ is normalized since only the direction of the light is considered. Applying the above equation for all pixels in the point cloud, we obtain the corresponding light map, which is a matrix with the same size as the point clouds. However, it should be noted that the light map is calculated based on the vertex map, while the vertex map in the training pipeline is only semi-dense, so the light map is also semi-dense and has 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.

See chapter 4 for more details on how to set up the dataset.

3.5.2 Trip-Net Architecture

We proposed a network that considers the vertex map, the light map, and the grayscale image for surface normal inference, which is called the Triple-Pipe-Gated Network (*Trip-Net*). The *Trip-Net* uses the GCNN architecture three times to perform the normal inference task, so we call it *Triple-Pipe-Gated*. The architecture is shown in Figure 3.7.

The network consists of three pipes combined with one main pipe and two side pipes. Each pipe deals with a different task. The Main Pipe deals with the geometry information, which takes the vertex map as input and is used to predict the surface normal. The light map takes a Side Pipe as input, from which the light features are extracted and then passed to the Main Pipe as additional information for normal estimation. The image map takes another Side Pipe to extract the image features, and then forwards the features to the Main Pipe as well. The additional pipes provide the illumination information that helps the Main Pipe refine the inferred normals.

Side-Pipe (Light) The structure of the Light-Pipe is almost identical to the architecture of the GCNN. It takes the light map $\mathbf{L} \in \mathbb{R}^{W \times H \times 3}$ as input and then passes through gated convolution layers three times to obtain the feature maps, $\mathbf{X}_{L1D} \in \mathbb{R}^{W \times H \times C}$, which has the same resolution as the input map. Then three

downsampling operations are followed afterwards. In each downsampling, 3 gated convolution layers are used to further extract the feature maps, where the first two gated convolution layers have stride (1,1) and the third gated convolution layer has stride (2,2). 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, each up-sampling is done in three layers, an interpolation layer based on the nearest neighbor algorithm, a concatenation layer to concatenate the interpolated feature maps and the corresponding feature maps in the down-sampling part, and a gated convolution layer to reduce the channel size. The light pipe branch is completed in the last layer of the third up-sampling.

In this branch, four kinds of feature maps $\mathbf{X}_{L4D}, \mathbf{X}_{L3U}, \mathbf{X}_{L2U}, \mathbf{X}_{L1U}$ are used as guided information for further operation.

Side-Pipe (Image) The task of the Image-Pipe in the network is to predict the image features. This pipe is a pipe cooperating with the Light-Pipe. The architect is the same as that of the Light-Pipe, but only the input is the image matrix $\mathbf{I} \in \mathbb{R}^{W\times H\times 1}$. The first set of the feature maps are $\mathbf{X}_{I1D} \in \mathbb{R}^{W\times H\times C}$, noting that it has the same channel as the light feature maps. Then, the down-sampling is performed three 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})

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

It requires three times upsampling to produce 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 upsampling part have the same architecture as the Light Pipe. In the Image Pipe, four types of feature maps $\mathbf{X}_{I4D}, \mathbf{X}_{I3U}, \mathbf{X}_{I2U}, \mathbf{X}_{I1U}$ are used as guided information for further operation.

Main-Pipe (Vertex) The task of the vertex pipe in the network is to predict the normal map directly, also taking into account the feature maps in the other channels.

The input is the vertex map $\mathbf{V} \in \mathbb{R}^{W\times H\times 3}$ converted from the point cloud. The downsampling part is still the same as for the other two pipes. The feature maps in each downsampling part are of the form $\mathbf{X}_{V1D} \in \mathbb{R}^{W\times H\times C}$, $\mathbf{X}_{V2D} \in \mathbb{R}^{\frac{W}{2}\times\frac{H}{2}\times C}$, $\mathbf{X}_{V3D} \in \mathbb{R}^{\frac{W}{4}\times\frac{H}{4}\times C}$, $\mathbf{X}_{V4D} \in \mathbb{R}^{\frac{W}{8}\times\frac{H}{8}\times C}$, respectively, (represented as a regression function i_{down})

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

The up-sampling operation has a different situation than the other two pipes. It merges the output feature maps with the corresponding resolution from three pipes to get a merged 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: An interpolation layer to double the resolution using the nearest interpolation algorithm, a gated layer to reduce the channel size to 1/3 of itself for inter-pipes feature reduction, a concatenation layer to connect the output with the corresponding feature map in the downsampling part (skip connection), a gated layer to reduce the channel size to 1/2 of itself for skip-connection feature reduction, a concatenation layer to merge the corresponding upsampling feature map from the other two pipes with the feature map in the current pipe altogether. These 5 layers take into account both the information from the other pipes and the feature maps from the downsampling part.

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

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

3.5.3 Loss Functions

We have investigated different loss functions for training our network. The most common loss functions are the L1 loss and the L2 loss for image simulation. In our experiments, we used a modified Huber loss function to train our model, which yields a lower error in the evaluation compared to a pure L1 or L2 loss function.

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 mean value of the absolute errors 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.

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

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

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

where $c = 0.2 \max(|\tilde{y} - y|)$. The shape of the loss is shown in Figure 3.8. Huber loss is a combination of L1 and L2 loss, being L2 loss in the range $[-c, c]$ and L1 loss outside this range. The loss is also differentiable at the junction of two first order intersection functions. As mentioned in [17], the parameter $c = 0.2 \max(|\tilde{y} - y|)$ corresponds to the 20% of the maximum error per batch.

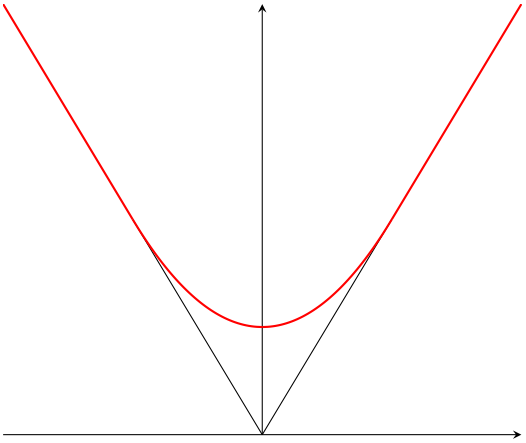


Figure 3.8: The shape of Huber Loss (show in red line).

Figure 3.5: The architecture of Gated convolution neural network (*GCNN*) based on Gated convolution and *U-Net* Architecture.

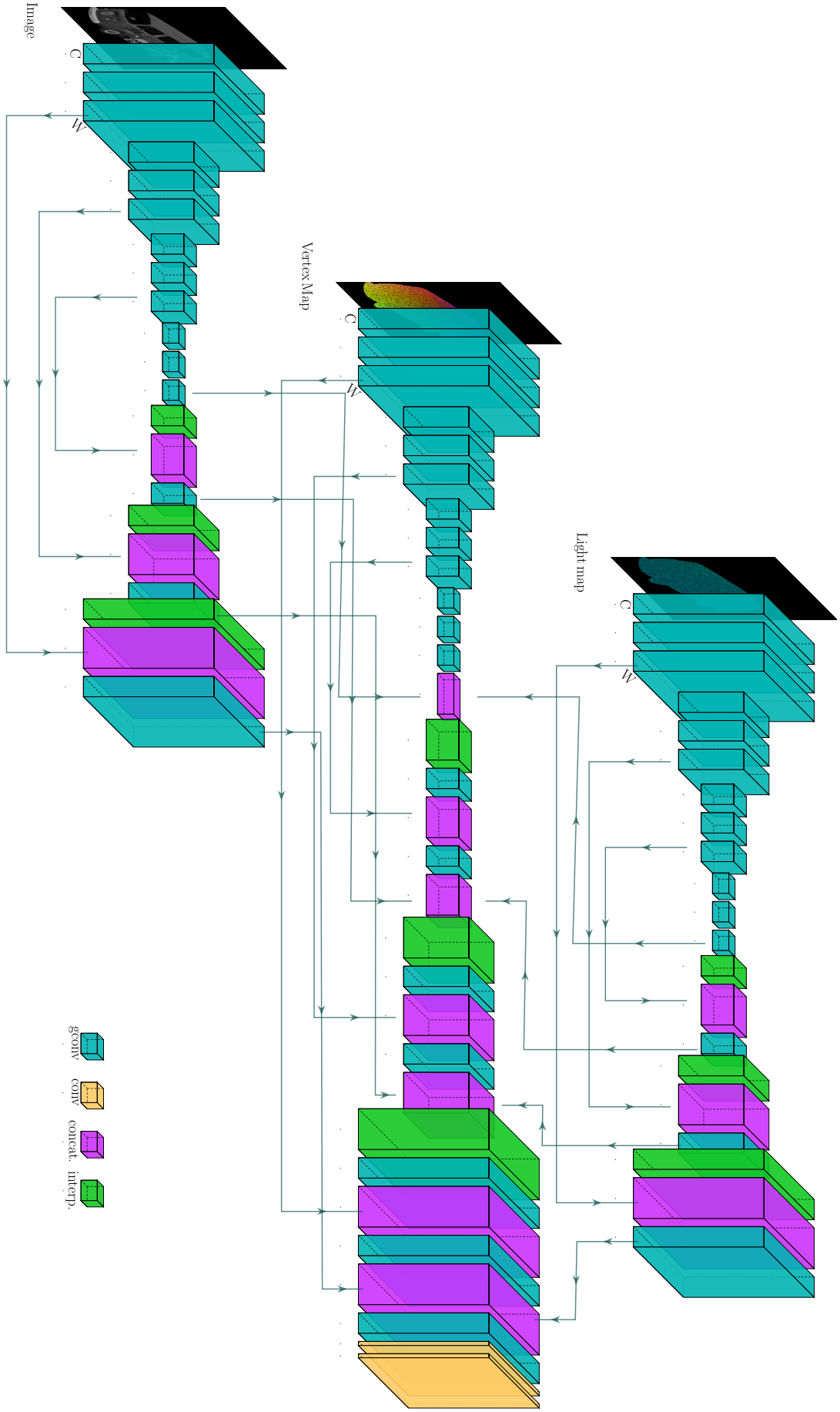


Figure 3.7: The architecture of the *Trip-Net*.

Chapter 4

Dataset

In this work, we need two types of input data to train our models. First, we need to know the geometry of the object surface. This can be captured by a depth camera, which records the distance of the object surface from the optical center of the camera and can be converted into a point cloud. Second, we need information about the illumination of the object surface, which is used as photometric information for further improvements. This type of information requires an observed image of the object with the light directions projected onto the object surface.

To obtain this data, we can use a structured light scanner to scan the objects in the laboratory. However, the scanner usually cannot scan the dark, shiny, and transparent areas, resulting in many missing pixels and holes in the depth map. This incomplete surface information makes it difficult to train our model because the ground truth corresponding to the depth map is also incomplete. Second, training a deep learning-based model usually requires a large dataset, especially for a network without a backbone. Creating a single deep map dataset for this work would require too much work and resources. A suitable way to obtain huge depth map data with illumination information is to use a game engine that can simulate an arbitrary number of data.

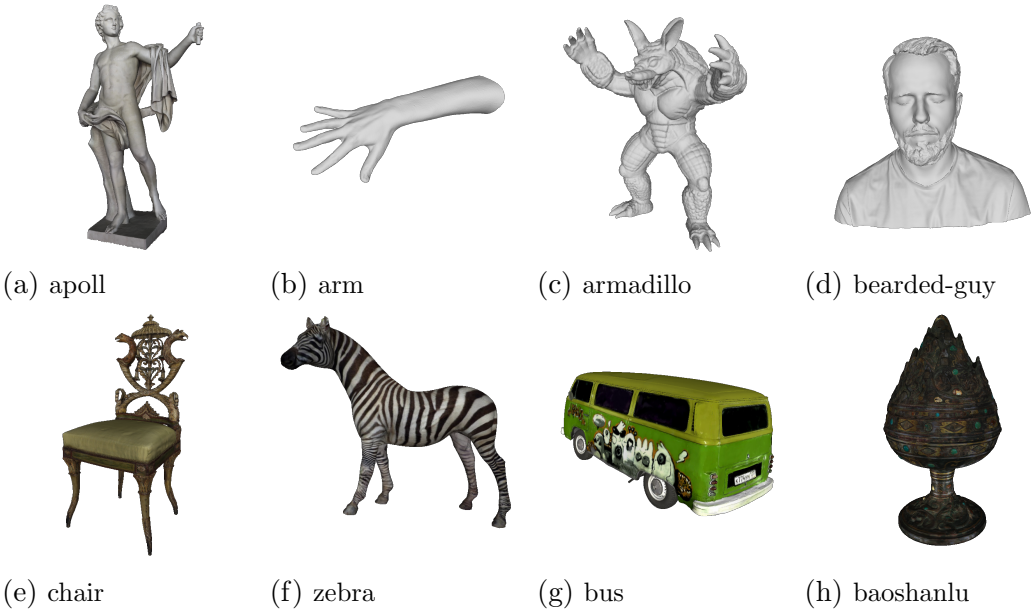
We use the Unity game engine for data generation. Using a *C#* script, we can set up a similar configuration in the lab. We then create a dataset based on the collected objects. The dataset we created for this work is called *synthetic-50-5* because it contains 50 different object models for training and 5 object models for testing.

4.1 Data Resources

The object models we used were collected from the Internet. A number of point cloud datasets for image processing research have been published by [31], [22], [21], and [29]. Some of these point clouds were scanned from real objects using high-resolution scanners such as the Cyberware 3030 MS+ and calibrated with post-processing. These objects were scanned hundreds of times, fully capturing the original objects with up to millions of points[31]. Some of them are fully digitally synthesized. The dense point clouds make the normal inference task trivial, as the neighborhood-based method works adequately for this type of task. Some of the point clouds even contain pre-computed normal maps based on more advanced methods. They all provide the accurate ground truth for the supervised learning method.

The *synthetic-50-5* is a dataset we created in this work based on 50 point clouds as the training set and 5 point clouds as the test set. In creating the dataset, we tried to use as many object types as possible to cover a robust and wide range of training scenarios. There are several categories of models, such as figurines, animals, statues, toys, furniture, antiques, and car models, some of which have relatively smooth surfaces, such as model *arm, bus, rabbit, cat, zebra*, and some of which have very intricate details, such as model *Washington, Car-Engine, Armadillo*. We created

this dataset for normal inference tasks. Figure 4.1 shows illustrations of some objects. Appendix A contains a complete version of the models of the dataset.



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

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

Figure 4.1: Some of the objects used for *Synthetic-50-5*

In addition, some objects also have colored textures, as shown in the second line of Figure 4.1. This is specifically prepared for the illuminated approach, since the image is used to evaluate the surface normals. By using textured models, our models get closer to the real dataset and have improved robustness that can be further applied to the real dataset.

4.2 Synthesizing Scenes using Unity

To simulate the data acquisition scenario as realistically as possible. We made the following settings. A flat cylinder, called *stage* , is placed in the center of the 3D space as a platform for placing objects. We fixed the *stage* in a predetermined position that will not be changed when images are captured. A directional light with RGB color FFF4D6 is placed 25 m away in the top view direction as an ambient light, which also has a fixed position. An RGB-D camera is placed about 10 m away from the stage in the top-view direction. The camera captures the depth map and the grayscale image, which are randomly arranged after each scene. The movement range of the camera is 0.1 m in both directions of the X, Y and Z axes and a randomly changed Euler angle of 1°. A point light is placed about 5 m away as the illumination light, which is also moved randomly after each scene. The range of motion is 0.3 m in both directions of the X, Y and Z axes and 1° randomly in the Euler angle.

During data acquisition, the object is randomly selected and placed on the stage. The object has a random rotation of up to 30° in the *roll* axis direction, a 30° rotation in the *pitch* axis direction, and a 180° degree rotation in the *yaw* axis direction. This gives the camera the ability to capture most directions of the objects. The layout in the Unity game engine is shown in Figure 4.2. We generate 3000 scenes with resolution 128×128 and 5000 scenes with resolution 512×512.

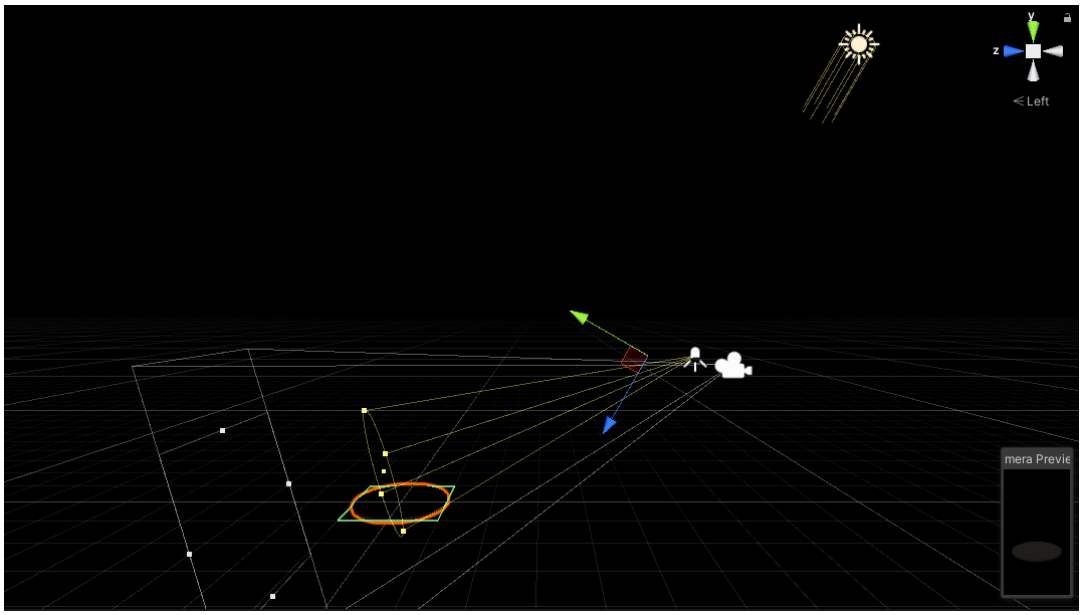


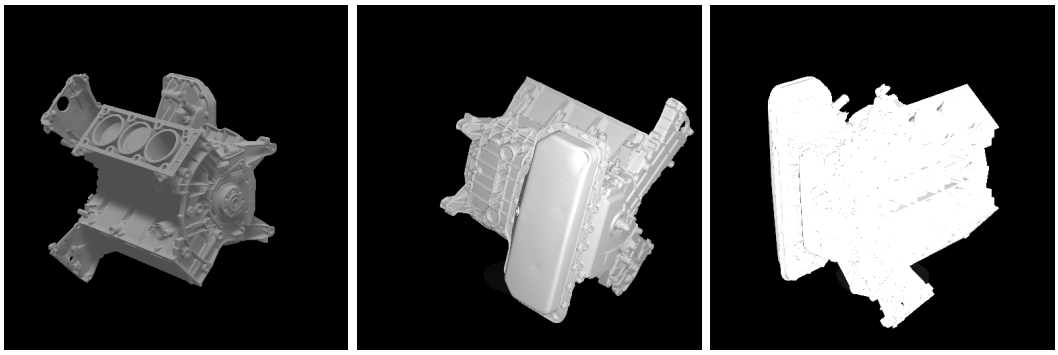
Figure 4.2: The layout of synthetic scene generation in Unity.

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

TABLE 4.1: The information saved for each scene in *synthetic-50-5*.

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

A critical detail we need to pay attention to is the exposure of the camera. Or we need to control the light emission on the object surface. As can be seen in Figure 4.3, too much exposure causes the surface texture to be difficult to see, resulting in clipping. In our experiments, we found that an appropriate light setting is essential for the illumination-based approach. Too much or too little lighting effects do not improve the illumination-based approach.



(a) Insufficient light (b) Just Right (c) Too much light

Figure 4.3: Different exposure to the objects.

4.3 Data Preprocessing

The raw data collected by Unity required further processing before it could be fed into the training models.

Depth Map A depth map is a 1-channel image that contains information about the distance from the object surface to the camera center. It is stored as a 16-bit grayscale image, meaning that each pixel is in the range $0 - 65535$.

The raw depth maps in real data acquired by light scanners usually have missing pixels. To achieve a good match to the real data, a uniformly distributed noise was added to the synthetic data to randomly remove the valid pixels in the depth maps. The simulated noise is uniformly distributed over the entire map with a certain noise intensity. A parameter μ is used to control the intensity of the noise. It specifies the μ pixel drop in percent. For example, at $\mu - 10$, 10% of the pixels are randomly removed. For each scene, the noise operation is based on a random μ in a range $[0, 50]$. In some scenes there are more missing pixels, in others less. The random noise intensity also allows the model to learn scenarios not only with noise, but also with low noise or even no noise. Figure 4.4 shows the noise effect at different μ.

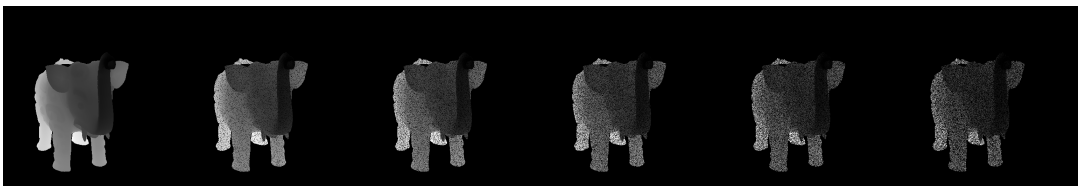


Figure 4.4: From left to right: Noise-intensity on $\mu - 0$, $\mu - 10$, $\mu - 20$, $\mu - 30$, $\mu - 40$, $\mu - 50$. Object Name: *elephant-zun-lid*.

The depth map is converted to a 3D vertex map after adding the noise. Consider a 3-dimensional Euclidean space. The X and Y axes are perpendicular to each other align with the direction of width and height of the depth map separately. The Z axis is point inward (i.e., from view point to the depth map). The depth provided in the depth map is the distance from camera center to the object surface. Thus, in order to find the corresponding 3D vertex map $\mathbf{V}_C$ with respect to the camera coordinate

system, we only need to multiple the depth matrix **Depth** with the point direction matrix **Dir**.

$$\mathbf{V}_C = \mathbf{Depth} \odot \mathbf{Dir}$$

For the direction $\mathbf{Dir}(x, y, z)$ of a point $\mathbf{V}(x, y, z)$, it's X and Y components can be mapped from the corresponding pixel position (u, v) on the depth map, whereas the Z component is the focal length fk in pixels. Then we have to further normalize it to an unit vector.

$$\mathbf{Dir}(x, y, z) = \frac{(u, v, fk)}{\|(u, v, fk)\|_2}$$

Conversion of a point from the camera coordinate system to the world coordinate system using the extrinsic matrix R and t

$$\mathbf{V}_W = \mathbf{V}_C R + t$$

The sizes of the individual training objects are different. We normalized them to a unit scale so that they have a relatively similar distance from the camera. In Figure 4.5 we have plotted the variations in value in each axis before normalization. Table 4.2 gives a quantitative evaluation of the corresponding average values.

The normalization was performed as follows. First, the points are translated to the original point as much as possible, then the range value of an axis is chosen as the scaling factor, and the points are normalized to unit vectors. The equation is represented as follows

$$\begin{aligned} X_n &= \frac{X - \min(X)}{s} \\ Y_n &= \frac{Y - \min(Y)}{s} \\ Z_n &= \frac{Z - \min(Z)}{s} \\ s &= \max(X) - \min(X) \end{aligned}$$

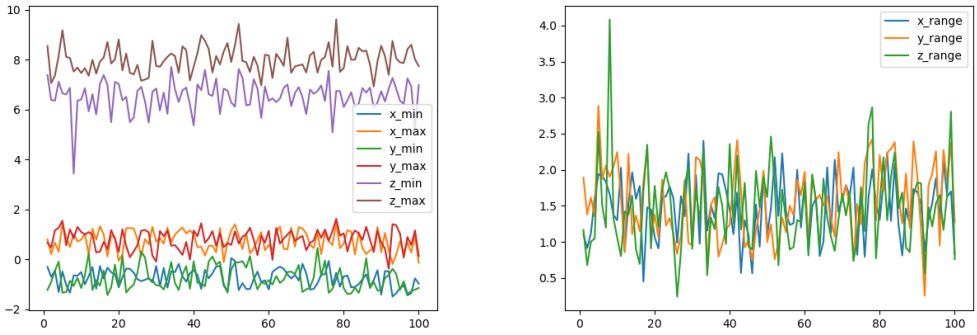
where s is a scaling factor calculated as the range of the X axis, but theoretically it can also be the range of Y or Z axis.

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.

Image The grayscale image can be used for the photometric stereo method and also as readable information for humans. Since the image captured by the camera is in RGB format, we need to convert it to grayscale to fit our models. This is done using the following equation.

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



(a) Extreme Values in each axis

(b) Vertex Range in each axis

Figure 4.5: The position fluctuation of the points in 100 Vertex Maps. Left: Extreme values in 3 axes; Right: Vertex range in 3 axis.

Normal Map The normal map is the tangential surface normal stored in an 8-bit/channel RGB image. The surface normal (n_x, n_y, n_z) and the corresponding RGB color (R, G, B) can be converted using the following equation:

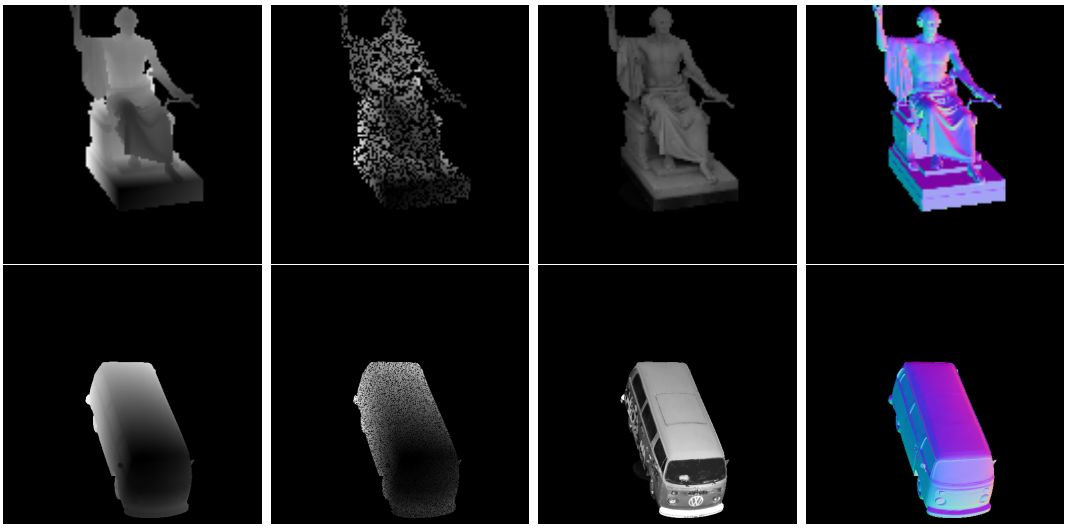
$$n_x = \frac{R}{255} \cdot 2 - 1$$
$$n_y = \frac{G}{255} \cdot 2 - 1$$
$$n_z = 1 - \frac{B}{255} \cdot 2$$

Chapter 5

Experiments

5.1 Training Details

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



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

Figure 5.1: Some of the evaluation scenes during the training. The objects from top to bottom rows: *Washington*, *Bus*

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

For the training parameters, we set the training pipeline with a batch size of 8, Adam Optimizer[14], learning rate starting at 1×10^{-3}, it will decrease at epoch 8 with learning decay factor 0.5. The model is trained using PyTorch 1.10.0a0, CUDA

11.4.1, GPU with an NVIDIA A100 Tensor Core GPU. We stop training when the angular error of the normal map stops decreasing.

The *GCNN* is the base model of the whole work. The architecture is described in 3.4. We use a single *GCNN* to estimate surface normals as a geometry-based approach. It uses a vertex map as input. To verify the applicability of the skip-connection and gated-convolution layers, we trained two additional models for comparison. In the first model, we replace all gated layers with default convolutional layers in the network, but keep all other settings and give it the name *CNN*. It is used to compare the performance between the gated layer and the default convolutional layer for our normal inference task. As mentioned in 3, the gated layer is designed to process noisy input. Since the entire vertex map in the dataset was noisy, the *GCNN* should outperform *CNN*. Another model called *NOC* is used to check skip connections, which simply removes the skip connections in the network but leaves the other settings unchanged. Its purpose is to show to what extent the performance of the model is improved by skipping connections. We use Huber Loss as a loss function during training. We have found that it gives a better final error compared to L2 loss.

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

TABLE 5.1: GCNN model information. The V-P, L-P, and I-P columns indicate the number of convolutional layers in the vertex pipe, light pipe, and image pipe, respectively. Note that a gated convolution layer consists of 2 standard layers and is therefore counted as 2.

The trip-net model uses a triple GCNN architecture with 4 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 the baseline to observe the advantages of the illuminated information with the trip-net architecture. We also examined the optimal fusion times of the Trip-Net to determine a possible simplification of the model. A set of similar models was trained with the same settings but different fusion times, denoted Trip-Net-Fx, where x denotes the fusion times. We score the fusion times from 1 to 4. For the learning rate, we initially set 0.001. This fits the GCNN model well, but leads to a loss explosion in the trip-net. Therefore, we set a learning rate schedule with an additional decay step at epoch 8. The decay factor is 0.5. The batch size is set to 8.

During training, we found that the trip-net with 4 fusions converges significantly faster than models with fewer fusion times. However, model F3 with three times fusion converges slower than F4, but ends up with a similar loss of score as F4 (see Qualitative Evaluation). Models F1 and F2 are relatively less accurate than the other two models, but the sacrifice of accuracy results in a relatively lighter model. Since they have fewer fusion times, the corresponding upsampling layers in the image and light pipes can also be removed. The model can be trained faster, and the size is also smaller. Table 5.2 contains a comparison of the size of different models.

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

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

5.2 Quantitative Evaluation on 6 Metrics

We evaluated our models on a synthetic dataset with a resolution of 128× 128. Based on the metrics proposed by [7], 6 different metrics are used for the evaluation. Note that the vertex map input is only semi-dense. One of the advantages of the GCNN architecture is that it is robust to noisy input, so all points, including missing points in the input vertex map, are considered in the evaluation. We use $\tilde{\mathbf{N}}$ for the estimated normal map, $\mathbf{N}$ for the ground-truth normal map, P for the *set* of valid points, K for the number of valid points. *median* is a function to calculate the median value of a given *set*.

Average Angle Error Metric (AAE) The metric calculates the average absolute angular error for each point between the inferred normal and the ground-truth normal map.

$$AAE = \frac{1}{K} \cdot \sum_{i \in P} \left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \cdot \frac{180}{\pi} \right|$$

Median Angle Error Metric (MED) The metric calculates the median of the absolute angular error of all points in the normal map.

$$MED = \text{median} \left(\left\{ \left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \right| \text{for } i \in P \right\} \right)$$

5 Degree Error Metric (E_5) The metric calculates the percentage of predicted normals that have an error less than 5° compared to ground truth. First, the number of normals that have an error less than 5° (D_5) is calculated.

$$D_5 = \sum_{i \in P} \left(\left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \cdot \frac{180}{\pi} \right| < 5 \right)$$

Then the percentage of normals that have error less than 5° (E_5) is calculated.

$$E_5 = \frac{D_5}{K} \cdot 100\%$$

11.5 Degree Error Metric ($E_{11.5}$) The metric calculates the percentage of predicted normals that have an error less than 11.5° compared to ground truth. First, the number of normals that have an error less than 11.5° ($D_{11.5}$) is calculated.

$$D_{11.5} = \sum_{i \in P} \left(\left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \cdot \frac{180}{\pi} \right| < 11.5 \right)$$

Then the percentage of normals that have error less than 11.5° ($E_{11.5}$) is calculated.

$$E_{11.5} = \frac{D_{11.5}}{K} \cdot 100\%$$

22.5 Degree Error Metric ($E_{22.5}$) The metric calculates the percentage of predicted normals that have an error less than 22.5° compared to ground truth. First, the number of normals that have an error less than 22.5° ($D_{22.5}$) is calculated.

$$D_{22.5} = \sum_{i \in P} \left(\left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \cdot \frac{180}{\pi} \right| < 22.5 \right)$$

Then the percentage of normals that have error less than 22.5° ($E_{22.5}$) is calculated.

$$E_{22.5} = \frac{D_{22.5}}{K} \cdot 100\%$$

30 Degree Error Metric (E_{30}) The metric calculates the percentage of predicted normals that have an error less than 30° compared to ground truth. First, the number of normals that have an error less than 30° (D_{30}) is calculated.

$$D_{30} = \sum_{i \in P} \left(\left| \arccos \frac{\mathbf{N}_i \cdot \tilde{\mathbf{N}}_i}{|\mathbf{N}_i||\tilde{\mathbf{N}}_i|} \cdot \frac{180}{\pi} \right| < 30 \right)$$

Then the percentage of normals that have error less than 30° (E_{30}) is calculated.

$$E_{30} = \frac{D_{30}}{K} \cdot 100\%$$

We evaluated our trained models on *synthetic-50-5*. In the test dataset, 5 objects are considered. They are: *Baoshanlu* (BL), *Bus* (BS), *Dragon* (DG), *Garfield* (GF) and *Washington* (WS). Each object has 20 scenes with a total of 100 scenes for 5 objects. The test objects are not present in the training dataset. We evaluate all the presented models on the test dataset, to fit them into a table, the names of each model are simplified. The model *SVD* uses the SVD optimization method, the model *NOC* is the no-skip connection version of *GCNN*, *CNN* is the CNN version of *GCNN*. *F1*, *F2*, *F3*, *F4* are the fusion times in the *Trip-Net*.

When evaluating the *GCNN* models, we take *SVD* model with an optimal neighborhood size as baseline. *NOC* and *CNN* are used to verify the performance of *GCNN* model. When evaluate the *F1-F4* models, we can take *GCNN* model as baseline.

From the table we can see that all the learning based approaches achieves a better result than SVD approach. In the 30 degrees error metric, GCNN based approach achieves 95 % accuracy whereas Trip-Net is even higher, some of the models like *dragon* achieves 98%. The best performance is around 90%, 75%, 45% in 22.5°, 11.5° and 5° degrees error metrics respectively. Another notable result is the close performance of F3 and F4 models, where they achieves a very comparable performance. We also found that during the training, F4 model converges faster than F3, but F3 in the end achieves a similar loss with model F4. However, based on the 30°, 22.5°, 11.5° metrics, we can still see that F4 model gives a more stable performance with less

high error normals. This is reasonable since the last fusion in the original resolution provides more high resolution information to the models.

In evaluating the *GCNN* models, we take the *SVD* model with an optimal neighborhood size as the baseline. *NOC* and *CNN* are used to check the performance of the *GCNN* model. When evaluating the *F1-F4* models, we can take the *GCNN* model as a baseline.

From the table, it can be seen that all learning-based approaches achieve a better result than the *SVD* approach. In the 30-degree error metric, the GCNN-based approach achieves 93.53% accuracy in average, while Trip-Net is 94.81%. Some models such as *dragon* achieve 97.61%. The Trip-Net achieves around 90%, 75%, 45% in the 22.5°, 11.5° and 5° degree error metric, respectively. We also found that model F4 converges faster than less fusion models during training. This is reasonable since the last fusion at the original resolution provides more high-resolution information to the models.

Obj.	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
BL	22.58	11.08	13.13	13.71	10.14	10.66	10.66	**10.09**
BS	19.78	7.17	8.67	11.71	**6.85**	7.52	7.93	7.09
DG	24.66	10.58	15.02	14.78	10.03	10.34	8.52	**7.58**
GF	23.87	9.82	12.26	14.70	9.82	9.65	9.88	**9.50**
WS	26.30	13.46	17.60	17.92	13.42	13.15	12.81	**12.51**

TABLE 5.3: Average angular error of the evaluation dataset. *SVD* Neighborhood size $k = 2$

Obj.	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
BL	14.80	8.74	10.29	11.36	**7.69**	8.27	8.37	7.83
BS	8.80	3.54	4.70	8.55	**3.01**	4.01	4.69	3.56
DG	15.42	7.53	10.68	11.85	6.74	7.47	6.17	**5.18**
GF	14.36	6.14	8.54	11.71	6.08	**5.86**	6.38	5.96
WS	17.16	7.45	11.19	12.70	7.43	7.19	7.28	**6.93**

TABLE 5.4: Median angular error of the evaluation dataset. *SVD* Neighborhood size $k = 2$

Obj.	**SVD**	**GCNN**	**NOC**	**CNN**	**F1**	**F2**	**F3**	**F4**
BL	19.30%	24.75%	20.45%	15.86%	**34.04%**	27.93%	26.46%	29.41%
BS	41.39%	62.28%	52.70%	24.30%	**65.98%**	59.50%	52.56%	62.08%
DG	17.68%	31.05%	18.02%	13.64%	36.64%	30.20%	40.26%	**49.04%**
GF	25.37%	43.56%	29.39%	13.78%	43.86%	**44.85%**	40.42%	44.09%
WS	27.35%	37.25%	27.56%	14.42%	36.15%	38.55%	37.76%	**39.15%**

TABLE 5.5: Percentage of error of less than 5 degrees of the evaluation data set. *SVD* Neighborhood size $k = 2$

Obj.	SVD	GCNN	NOC	CNN	F1	F2	F3	F4
BL	40.82%	63.27%	54.23%	49.93%	66.76%	65.36%	65.12%	**68.29%**
BS	54.04%	83.23%	79.37%	66.58%	**83.62%**	82.43%	82.04%	83.15%
DG	40.87%	69.75%	52.93%	48.19%	72.64%	71.07%	77.83%	**81.52%**
GF	43.67%	73.48%	63.11%	48.81%	73.28%	74.21%	72.98%	**74.48%**
WS	40.47%	62.63%	50.17%	44.74%	62.65%	63.73%	63.88%	**65.07%**

TABLE 5.6: Percentage of error of less than 11.5 degrees of the evaluation data set. *SVD* Neighborhood size $k = 2$.

Obj.	SVD	GCNN	NOC	CNN	F1	F2	F3	F4
BL	62.27%	90.02%	84.28%	83.93%	90.68%	90.59%	90.96%	**92.04%**
BS	68.93%	93.26%	91.39%	88.58%	93.25%	93.22%	93.26%	**93.47%**
DG	60.71%	90.11%	79.72%	82.45%	90.33%	90.99%	93.95%	**94.78%**
GF	62.55%	89.83%	86.63%	83.47%	89.90%	90.18%	90.46%	**90.72%**
WS	56.92%	81.03%	71.81%	73.11%	81.13%	81.72%	82.68%	**83.37%**

TABLE 5.7: Percentage of error of less than 22.5 degrees of the evaluation data set. *SVD* Neighborhood size $k = 2$.

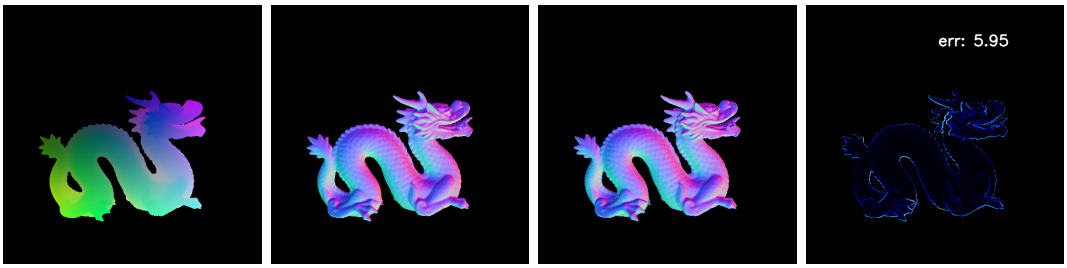
Obj.	SVD	GCNN	NOC	CNN	F1	F2	F3	F4
BL	73.34%	95.92%	92.88%	93.20%	96.28%	96.06%	96.51%	**96.85%**
BS	75.14%	95.80%	94.38%	93.22%	95.74%	95.76%	95.86%	**96.00%**
DG	68.72%	94.69%	87.86%	90.61%	94.61%	95.10%	97.15%	**97.61%**
GF	70.28%	93.76%	91.94%	90.96%	93.75%	94.04%	94.26%	**94.29%**
WS	64.51%	87.48%	80.57%	82.72%	87.80%	87.90%	89.02%	**89.30%**

TABLE 5.8: Percentage of error of less than 30 degrees of the evaluation data set. *SVD* Neighborhood size $k = 2$.

5.3 Visual Evaluation on SVD

The *SVD* approach can predict the normal map well if the given point cloud is dense. As shown in figure 5.2. It can successfully predict the smooth surface of the dragon object, especially the flakes and the tails of the dragon.

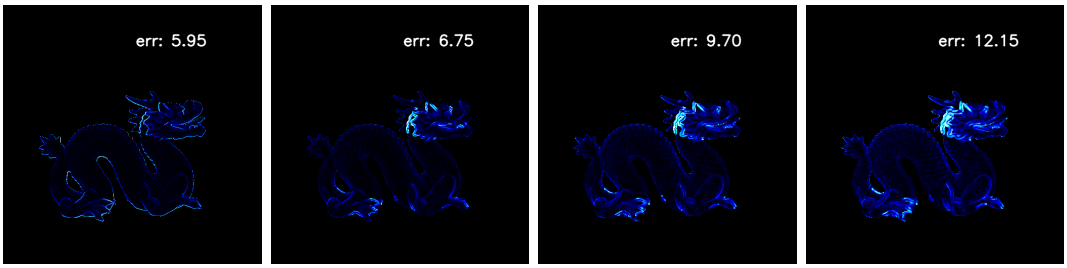
However, it fails in the areas such as the hind leg, horn, and mouth, which are mainly composed of sharp edges. This is because the neighboring points in these areas do not hold well to the coplanarity assumption, the normals of these neighbors can be very different.



(a) Vertex (b) GT (c) SVD (d) Error

Error: 0 90

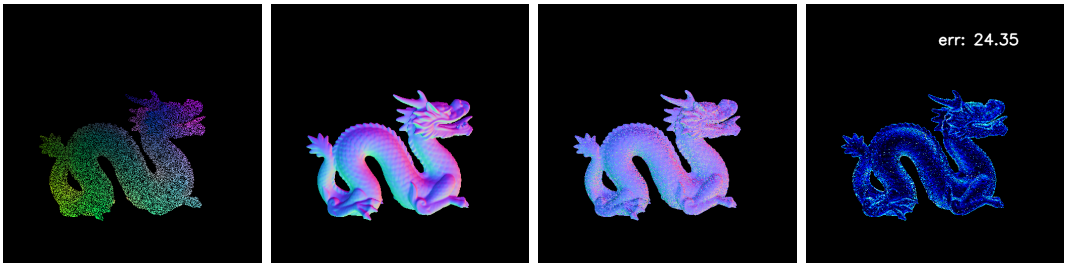
Figure 5.2: Normal map of a dragon object predicted by *SVD*. k=1. (Resolution: 512×512)



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$ (d) $k = 4$

Error: 0 90

Figure 5.3: Error map of *SVD* with different k values. (Resolution: 512×512)



(a) Vertex (b) GT (c) SVD (d) Error

Error: 0 90

Figure 5.4: *SVD* visual evaluation on a noised dragon model, $k = 1$, noise factor μ=50. (Resolution: 512×512)

The SVD approach depends on a well-chosen neighborhood size k. Figure 5.3 shows the evaluation for different k values. For $k = 1$, the average angle error of the whole image is the smallest, most of the normals are close to the ground truth, but the outline edges, i.e., the areas where the surface normals have changed severely. In the $k = 2$ case, the sharp edges are smoother and cause more error, such as the eye area of the dragon. Compared to the first case, the outline edge error is better. Most of the edge errors are reduced at $k = 2$ because more neighbor points are included in the evaluation and the effect of outliers is reduced. However, in the area of the horn

outline and the hindleg outline, the error becomes worse. In this case, most of the neighbors of these points are outliers. With $k = 3$ and $k = 4$, the average angular errors increase further compared to $k = 2$.

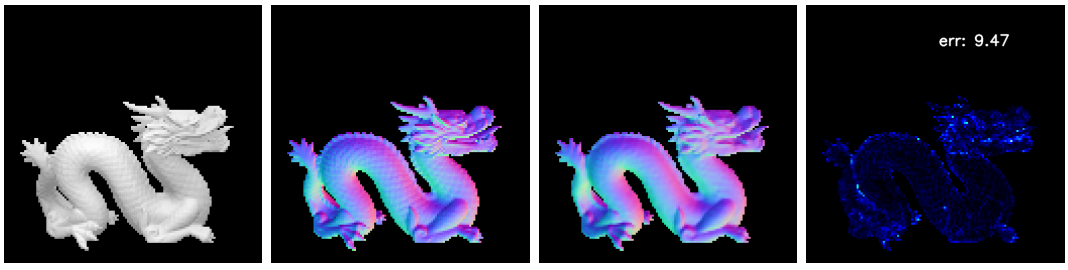
The performance of the neighborhood-based method is good enough for a well-chosen k. However, in the case of a noisy point cloud as input, this approach fails because the noise does not satisfy the neighborhood assumption and also reduces the number of possible neighbors of each point for a fixed k. See figure 5.4.

5.4 Visual Evaluation on GCNN

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

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

A qualitative evaluation of the object *Dragon* is shown in figure 5.5. As can be seen in the figure, the GCNN model achieves an average angular error of 9 degrees for this object. The image shows an overall good performance for the whole object. A more detailed evaluation can be seen in Figure 5.6, the normal accuracy is especially good on the smooth surface, such as the body area. In the same case, as shown in Figure 5.7, the NOC model has an overall worse normal than the GCNN model in the smooth area. The CNN model retains the skip connection and thus yields a sharper result than the NOC model, however, the smooth region of the model is still worse overall than the GCNN model. In addition, the CNN model yields a much brighter error map (implying a higher angular error) in the sharp regions such as the hind leg and the head region of the dragon object. Figure B.1 shows further evaluation of the GCNN model on other objects. The GCNN model gives a good result, but for the *Washington* model there is still a lack of sharpness in the detail areas such as the face and clothing.



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

Error: 0 90

Figure 5.5: GCNN visual evaluation. (Resolution: 128×128)

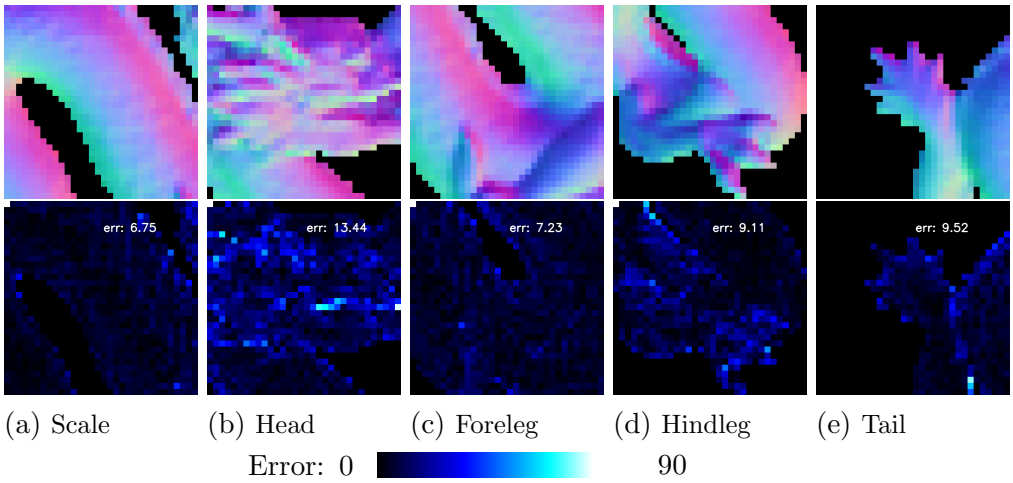


Figure 5.6: Zoom in of some regions of Dragon object based on GCNN. (Resolution: 32×32)

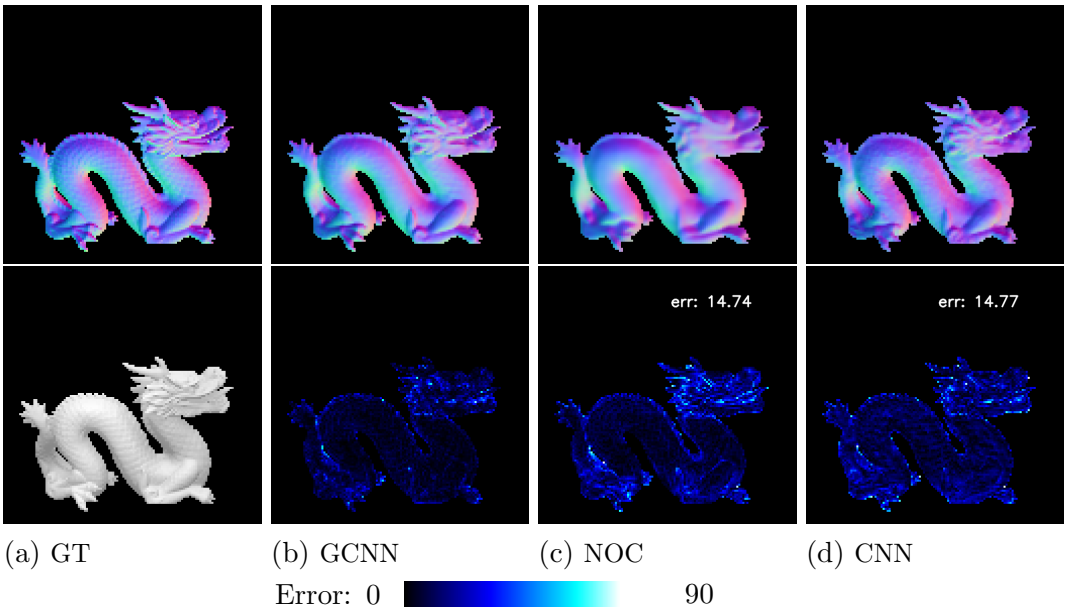


Figure 5.7: Comparison of GCNN based models. (Resolution: 128×128)

5.5 Discussion on the Feature Maps in GCNN

As shown in the figure, we visualize the feature maps of the last gated convolution layer in the first upsampling part for the models *GCNN*, *NOC* and *CNN*, which correspond to 128 feature maps with size 32×32 in width and height. The input data is also the same dragon object we used for the visualization. For each feature map, we assign the minimum value to the color on the far left of the color bar and the maximum value to the color on the far right of the color bar. We will now discuss the differences between these models.

To make the result clear, we use the green hue to visualize the feature maps and the brightness to distinguish the value scale: White corresponds to 1, Black corresponds to 0, and the colors with green hue correspond to some values in between.

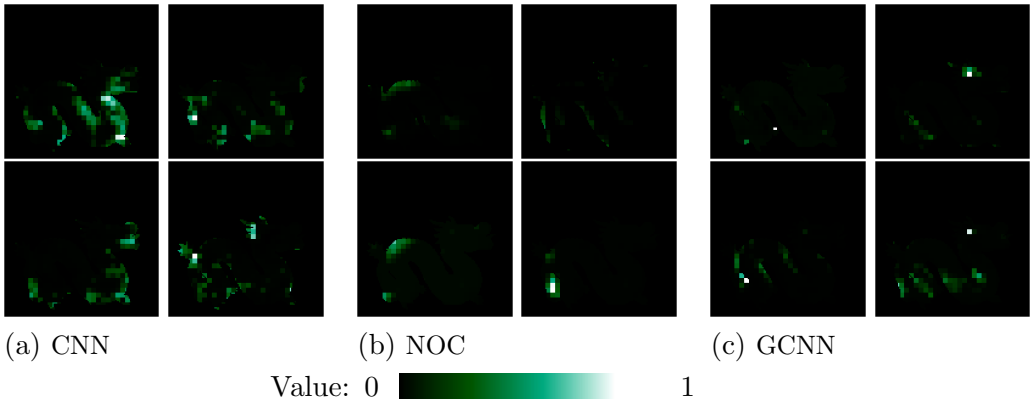


Figure 5.8: Some of detail feature maps of CNN, NOC, GCNN models in up-sampling part on resolution 32×32

We roughly distinguish feature maps into two types. The first type of feature maps learns the details of the target object that correspond to the feature map with sparse high values. In our example, the detail areas are the tail, head, paw, etc. Using the CNN feature maps, we could see that several detail areas in a feature map have high values. For example, if we observe one feature map in Figure 5.8 (a), the values in the tail, hind leg, and front leg areas of the dragon object are especially high than other areas. These high valued areas are exactly the detailed areas and the areas with higher errors in the evaluation. These feature maps show that the CNN model assigns high value to several detailed parts in a single feature map. In the case of the NOC model, one detailed part instead of several detailed parts is usually highly valued in a single feature map, as shown in Figure 5.8 (b). We can see that the hind legs are light, while other detailed areas such as the head or tail are dark. The NOC model uses one feature map to deal with only one specific detailed area. The GCNN model follows the patterns we found in NOC, each feature map has only one specific area that is very bright, and the rest of the areas are relatively dark. In addition, the bright area in the GCNN model is more concentrated than in the NOC model because of the skip connection, as can be seen in Figure 5.8 (c).

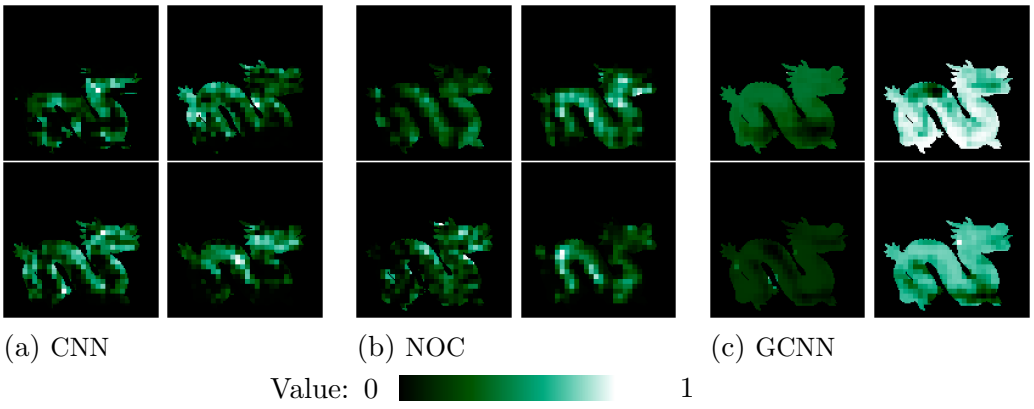


Figure 5.9: Some of global feature maps of CNN, NOC, GCNN models in up-sampling part on resolution 32×32

The second type of feature maps learns the global features of the target object corresponding to the feature map with dense, uniformly bright values. This type of feature maps provides an overview of the objects for rough prediction. As can be seen in Figure 5.9, the global feature maps have much bigger bright areas compared to the detailed feature maps. In the CNN model, the bright areas are seen in most parts of the object, but usually one or more specific areas are missing, such as the front or back legs. The global feature maps in the NOC model are either incomplete or blurry, so the global features are not learned properly. The GCNN model provides more complete global feature maps. The outline of the dragon is particularly easy to see compared to the other two models.

A complete version of the feature maps in this layer is shown in Figure C.1 in Appendix C.

5.6 Visual Evaluation on Trip-Net

In the approach that uses an illuminated, calibrated RGBD image, the task is performed by the Trip-Net presented in 3.5. Based on illumination and geometry information, the Trip-Net achieves a sharper and more accurate result compared to the GCNN model.

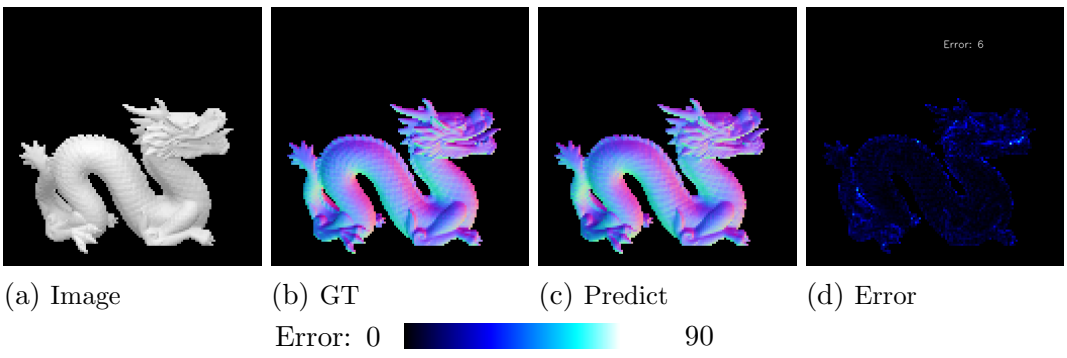


Figure 5.10: Trip-Net visual evaluation on resolution 128×128

The qualitative evaluation is shown in Figure 5.10. To show the effectiveness of the added illuminated information, the training settings are exactly the same for all

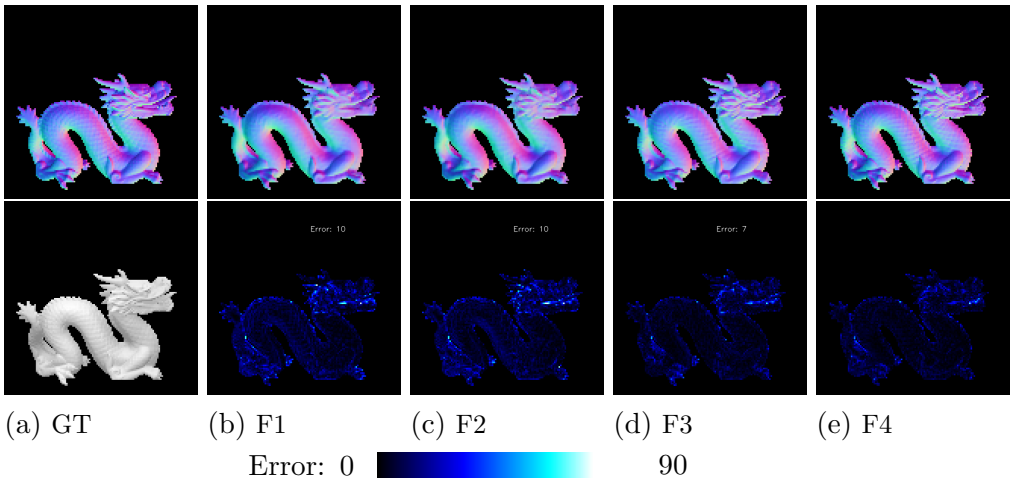


Figure 5.12: Comparison between different fusion times on Trip-Net Architecture (128×128).

models. We also use the same inputs as for the GCNN evaluation. The error of the Trip-Net is 6°, and that of the GCNN is 9°.

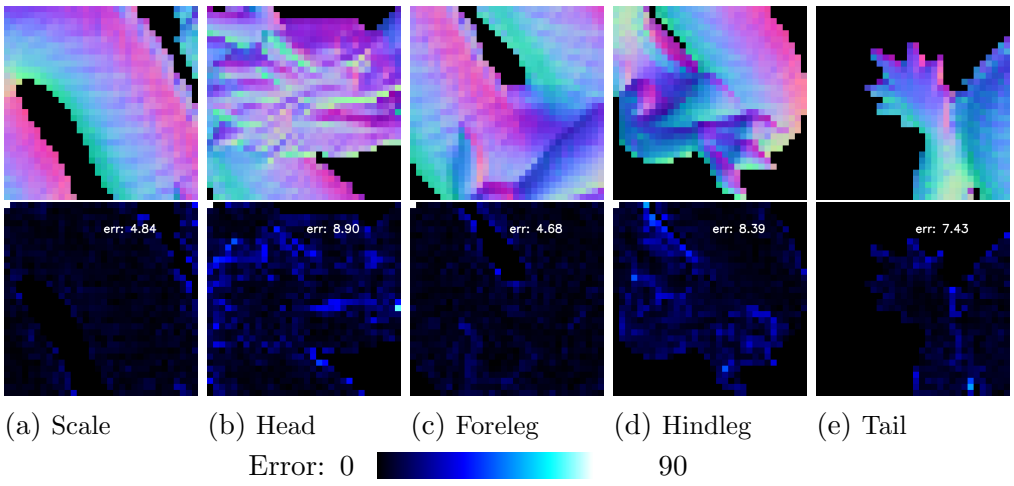


Figure 5.11: Zoom in of detail regions (32×32) of *Dragon* object (Trip-Net).

As can be seen in the Figure 5.11, the scale of the dragon's body is much easier to see and is also closer to the ground truth. The head region provides a sharper edge prediction. All five sampled zoom-in regions in Trip-Net have better performance than GCNN.

Figure 5.12 compares different fusion times on the Trip-Net model. It can be seen that the F1 with one fusion and F2 with two fusions models do not perform better compared to the GCNN model, which means that the illuminated information does not work when we consider only the lower resolution feature maps. The F3 model with three fusions and the F4 model with four fusions provide a much better result and also outperform the GCNN model. In these two fusion models, the high-resolution features are included. We can observe the dragon scale in the figures, it is much sharper in F3 and F4.

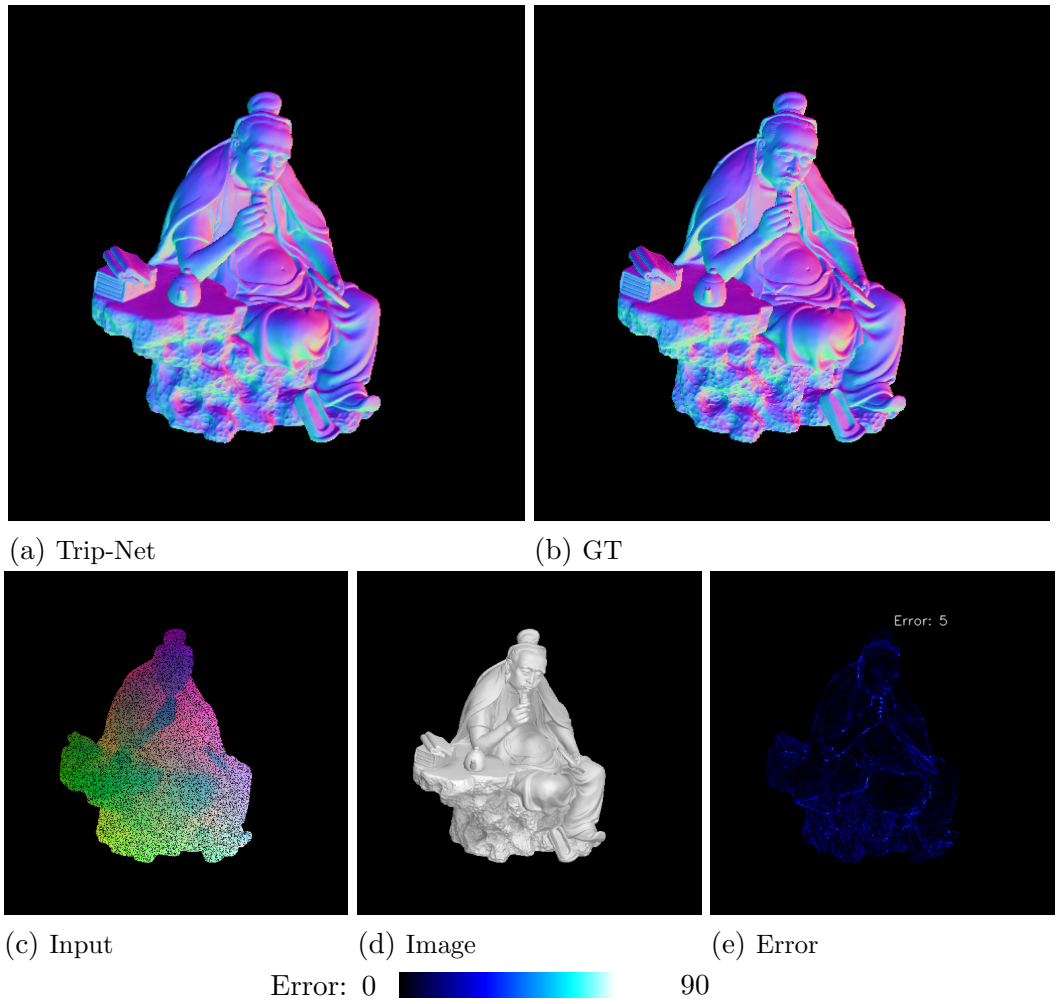
5.7 Trining on Higher Resolution

We also trained our models on a higher resolution dataset with 512×512 in width and height for each scene. Higher resolution data provides more information about surface features using the same model compared to lower resolutions. The advantage is that if we extract a fixed size patch from the normal map, say 32×32, it may correspond to a hind leg of the dragon object at a resolution of 128×128 scene, but at a resolution of 512×512 scene, it may only correspond to a toe. Thus, if we still use the same kernel size in the network for the same data set (3×3), the higher the resolution, the smaller the corresponding surface area. Thus, for the higher resolution, the surface becomes smoother and the surface normal can be computed more easily in a fixed size of area compare to lower resolution. This is a good thing. Because then we may only need to use these 32×32 points to calculate a toe in an image with 512×512 resolution. But in an image with a resolution of 128×128, the same area 32×32 could correspond to the entire hind leg of the dragon object. In other words. Thus, the higher resolution helps the model to compute a more accurate normal map.

Metrics	**SVD**	**GCNN**	**Trip-Net**
Mean	8.88	5.82	**5.33**
Median	**3.66**	3.98	3.67
5°	63%	63%	**66%**
11.5°	79%	89%	**91%**
22.5°	89%	**97%**	**97%**
30°	92%	98%	**99%**

TABLE 5.9: High resolution dataset evaluation of SVD, GCNN and Trip-Net models on 6 different metrics based on 100 test scenes. (Resolution: 512×512)

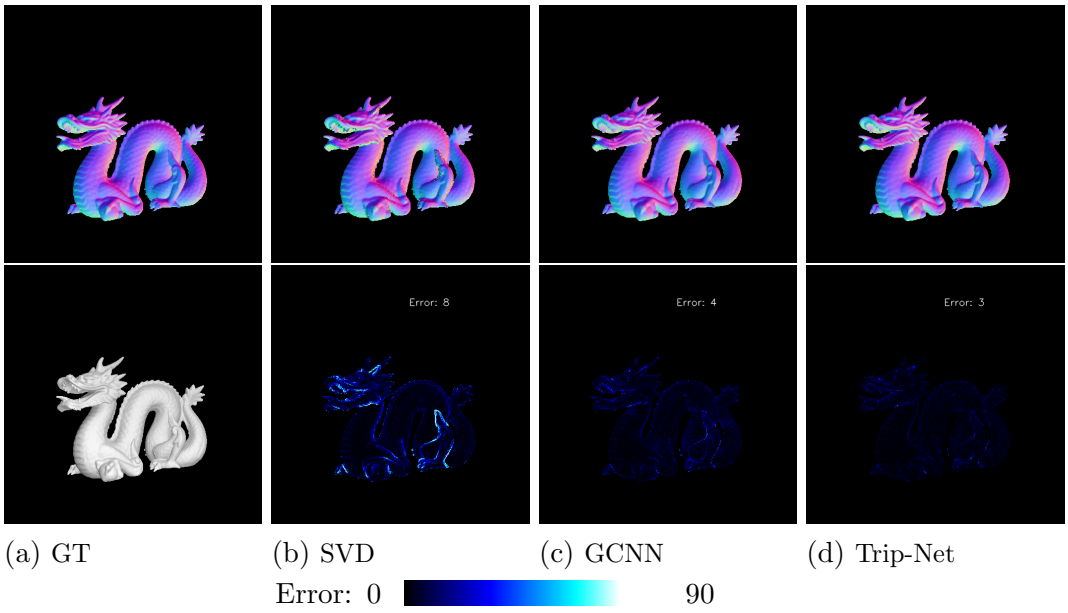
Since our model is a fully convolutional network, the architecture remains exactly the same on the high-resolution dataset. We use the same settings and the same model for training on a 512×512 resolution dataset. Training with the high-resolution network takes longer, but results in a lower angular error.



(a) Trip-Net (b) GT

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

Figure 5.13: Trip-Net visual evaluation on high resolution dataset 512×512.



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

Figure 5.14: Comparison on high resolution dataset (512×512).

A quantitative evaluation is shown in Table 5.9. It follows the same performance ranking compared to lower resolutions, i.e. GCNN is better than the SVD approach and Trip-Net is slightly better than GCNN. The accuracy is 99% in the 30° metric. In the average error metric, the error is 5°. A qualitative evaluation is shown in Figure 5.13. We use a figure object with smooth regions (belly and arm) and highly detailed regions with sharp surfaces (the uneven surface of the stone table) for the evaluation. Our models yield a 5° in the middle degree of the metric.

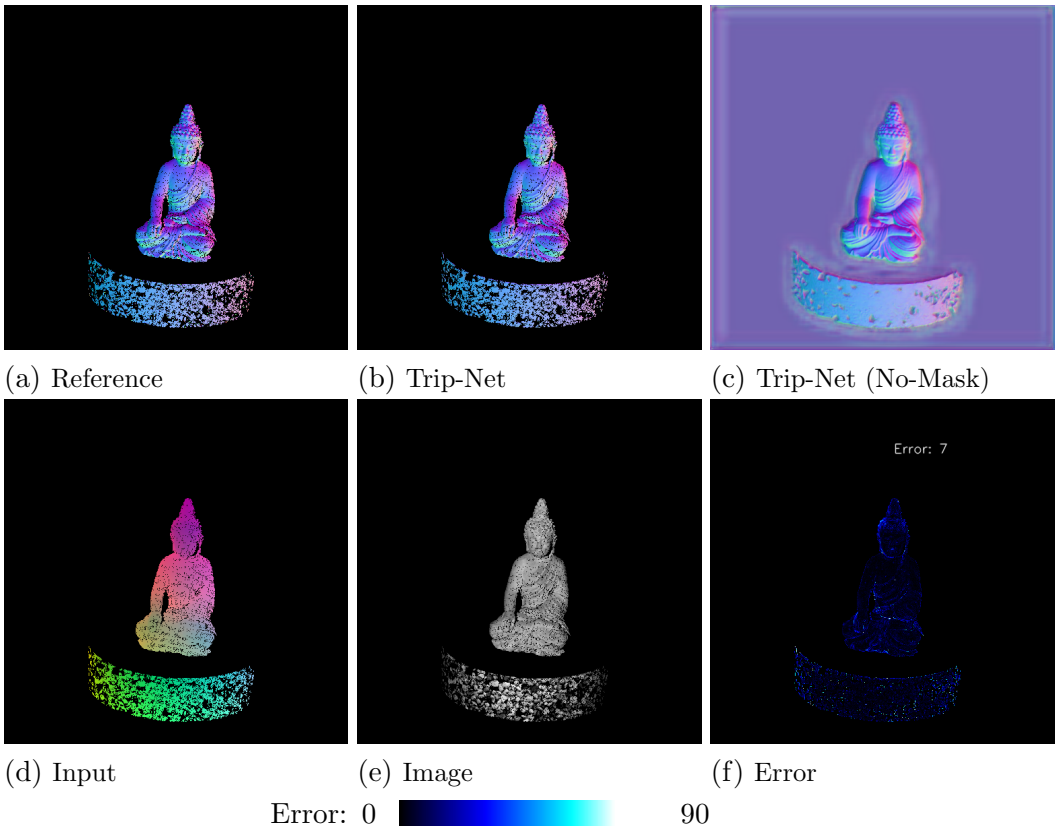
A comparison with other models is shown in Figure 5.14. Note that we use the same dragon object (with slightly shifted location since they were collected at different times in the high-resolution dataset) for the assessment to allow a coherent comparison with the low-resolution dataset. The *SVD* approach yields a good result (8.88° in the average degree metric). As mentioned earlier, the surfaces are relatively smoother when we keep the same window size for normal inference. In the high-resolution scenes, the percentage of missing pixels in a fixed window is the same as in the low-resolution scenes, but the remaining valid pixels in the same window size are on a relatively flat plane and are therefore good enough for accurate normal inference. However, as expected, we can still see the high-level error in the sharp edges of the dragon object, such as the horns and the hind leg areas, as shown in Figure 5.14.

5.8 Training on Real-Dataset

We also applied our model to a dataset acquired with a structured light scanner in our laboratory to test the applicability of our model to the real dataset. For the approach based on geometry information, we directly use the GCNN model trained on synthetic datasets since it only needs the depth map as input and the scenarios of the two datasets are the same. For the approach based on illuminated calibrated RGB-D images, the model needs to be refined based on the real dataset because the light intensity, position, and camera matrix are different. We have refined the Trip-Net based on a pre-trained GCNN model with the same settings in previous experiments.

Metrics	**SVD**	**GCNN**	**Trip-Net**
Mean	8.20	8.74	**8.09**
Median	**4.87**	5.70	5.00
5°	**51%**	44%	50%
11.5°	79%	79%	**81%**
22.5°	93%	93%	**94%**
30°	96%	96%	96%

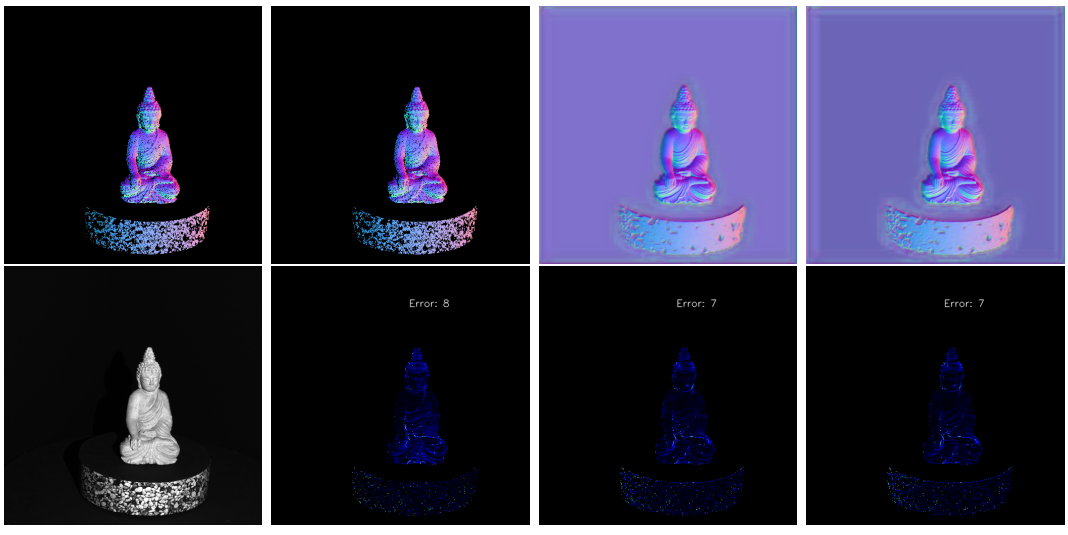
Table 5.10: Evaluation of SVD, GCNN and Trip-Net models on 6 different metrics based on 100 test scenes in Real-Dataset.



(a) Reference (b) Trip-Net (c) Trip-Net (No-Mask)

(d) Input (e) Image (f) Error

Figure 5.15: Real Dataset (512×512) evaluation (Trip-Net)



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

Figure 5.16: Real dataset (512×512) comparison.

We do not have a ground truth for evaluation in the Real-Dataset but we provide a normal map that calculated from another approach as reference. If we look directly at the visualization in Figure 5.15, we can see that the Trip-Net approach is well

able to “enhance” the missing pixels in the scenes and also gives a sharp result. The wrinkles on the clothes are detectable and even countable. Figure 5.16 compares the trip net with other approaches.

However, we also noticed that the large holes in the base level are preserved in the Normal Maps. These large holes correspond to the shiny and dark texture regions that are often present in the depth map captured by the scanners. Since these large holes are only related to specific feature areas and also have irregular shapes, we did not simulate this type of noise in the synthetic dataset, but only with a sparse binary mask. Therefore, our models could not fill in missing large holes in the estimated normal map. Further work on this topic could be to find a way to generate a very similar depth map noise to obtain a more robust training dataset.

Chapter 6

Conclusion

In our approaches, two networks have been proposed. The first network called GCNN is used for the geometry information based approach, which is our baseline method. It is particularly well suited for semi-dense input data, which is supported by the gated convolution layer design. With this network, we have achieved more accurate normal inference than with a standard layered network based on the same architecture. We also show the effectiveness of skip connectivity in the U-Net. In addition, the full convolution design allows us to use images with different resolutions as input.

The second network is called Trip-Net and is based on the GCNN model. It is designed for illumination-calibrated RGB-D image-based approaches. This model uses the GCNN architecture three times to process vertex map, light map and image separately. We also found that 4 fusions of the 3 pipes in our models is the key point to improve the performance. Based on our proposed dataset with resolution 128×128, the experiments show that the models trained on illuminated calibrated RGB-D images improve the accuracy by 10.4% of the average degree error compared to the geometry information based approach, which provides more sharpness in the normal map.

We also studied the performance of our models on a real dataset and compared it with traditional methods such as the SVD-based approach. In particular, our gated convolution layer-based model can fill in the missing pixels in the real input data and maintain an accurate normal estimate, while the optimization-based approach such as SVD cannot.

We collected 55 high-resolution 3D models from the Internet and used them to create our data. Specifically, we use an RGB-D camera, an ambient light, and a directional light to collect thousands of scenes. The scene generation is simulated by the Unity game engine. In addition, we also simulated noise in our synthetic dataset using an uniformly distributed drop to fit our model and adopt it with semi-dense input data. However, for the large missing regions, we did not generate similar noise in the training set, which results in our model not being able to fill the holes in the real data. Exploring an algorithm to mimic sensor noise to generate more realistic data for training can be a direction for further work.

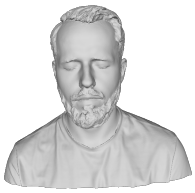
Appendix A

Dataset

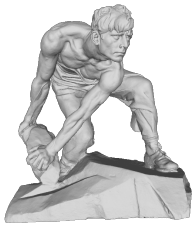
(a) apoll

(b) arm

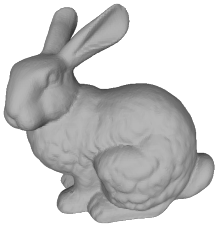
(c) armadillo



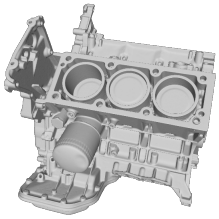
(d) bearded-guy



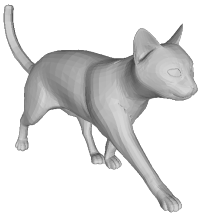
(e) bronze-sculpture



(f) bunny



(g) car-engine



(h) cat

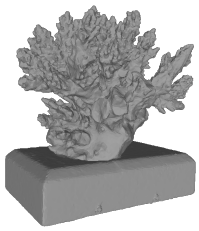
(i) side-chair

(j) chair2

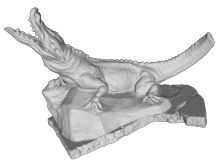
(k) chandelier

(l) Christmas-bear

(m) classic-side-table



(n) coral



(o) crocodile-statue

(p) Diane

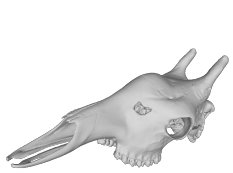
(q) Dresdner-knabe

(r) fantasy-dragon

(s) fox-skull

(t) Garuda and Vishnu

Figure A.1: Point clouds in dataset A



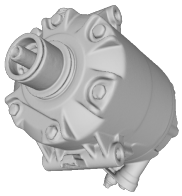
(a) giraffe-skull

(b) green-hat

(c) happy-budda

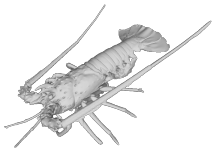
(d) heart-pendant

(e) indonesian-statue

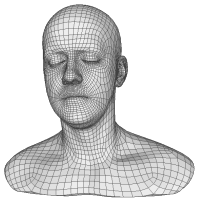


(f) industrial-compressor

(g) kylix



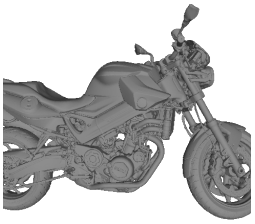
(h) lobster



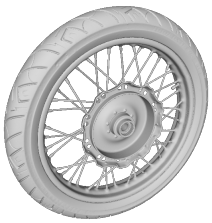
(i) head

(j) Lu-Yu

(k) Maitreya

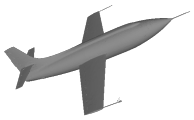


(l) motorbike



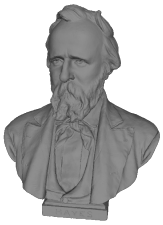
(m) motorcycle-wheel

(n) Pan



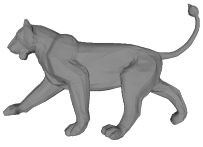
(o) plane

(p) elkins-refrigerator



(q) Rutherford

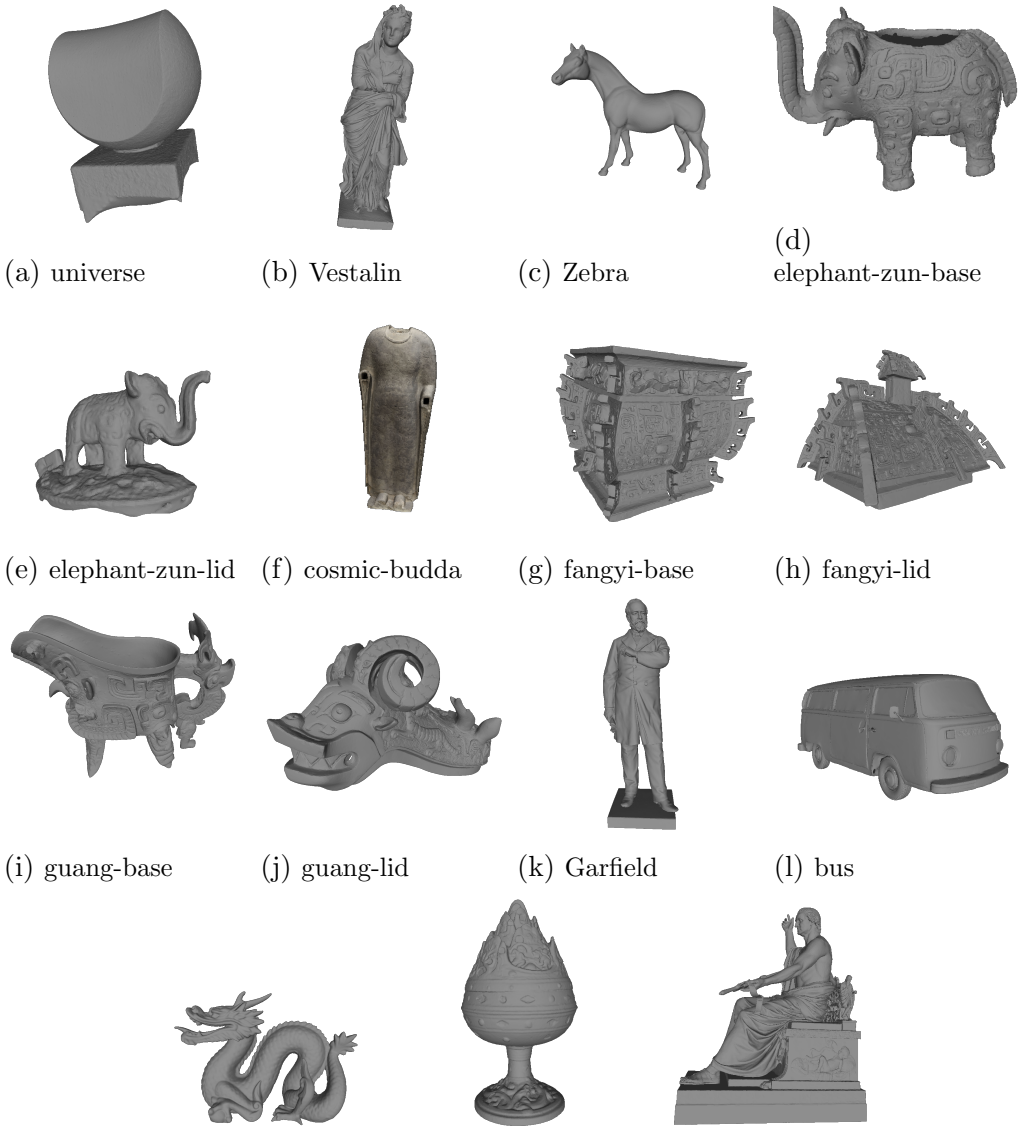
(r) statue-dragonfly-tamer



(s) Tiger

(t) transmission

Figure A.2: Point clouds in 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 dataset C

Appendix B

More Visualization

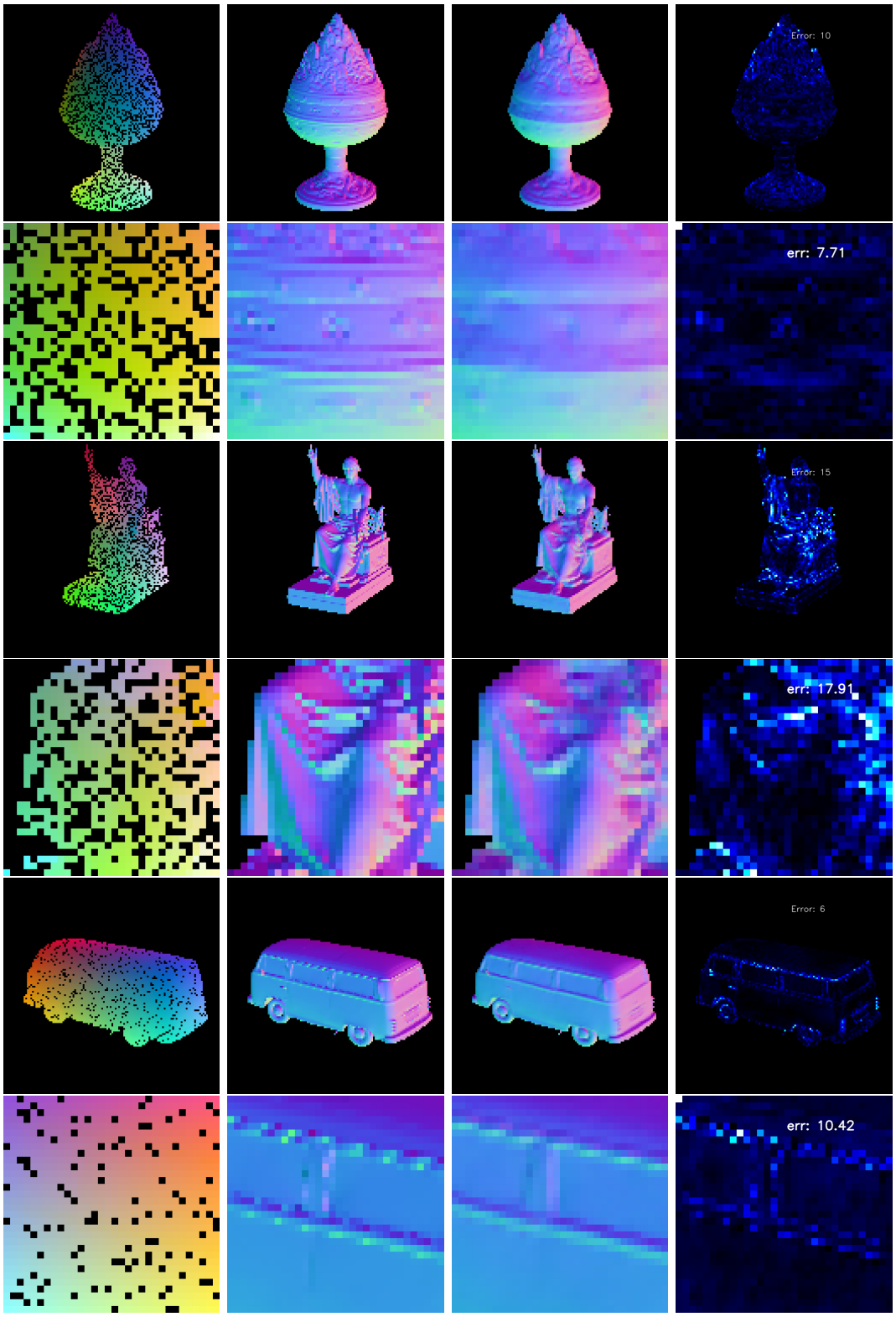


Figure B.1: GCNN evaluation (128×128). From top to bottom: Baoshanlu, Washington statue, Bus.

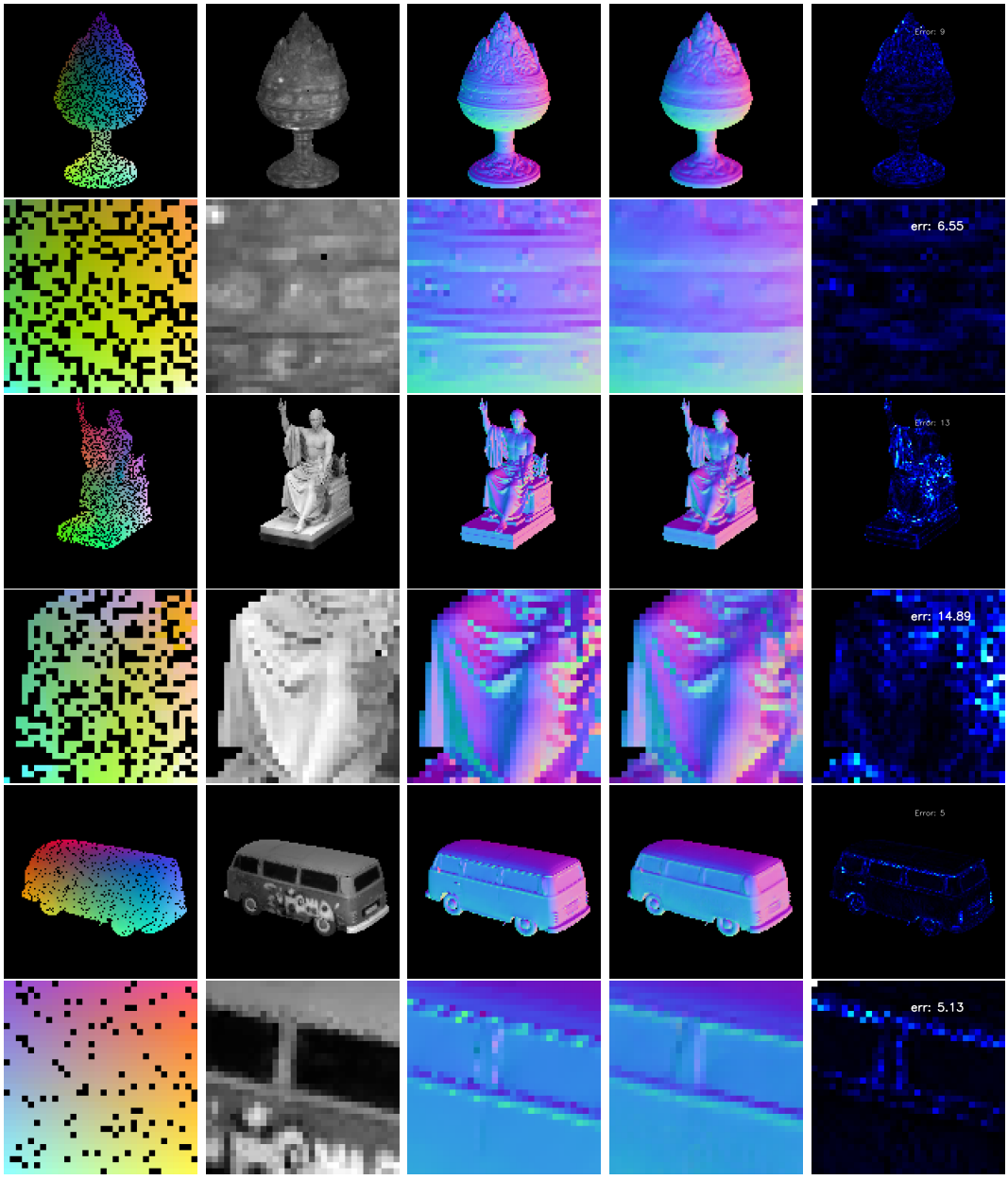
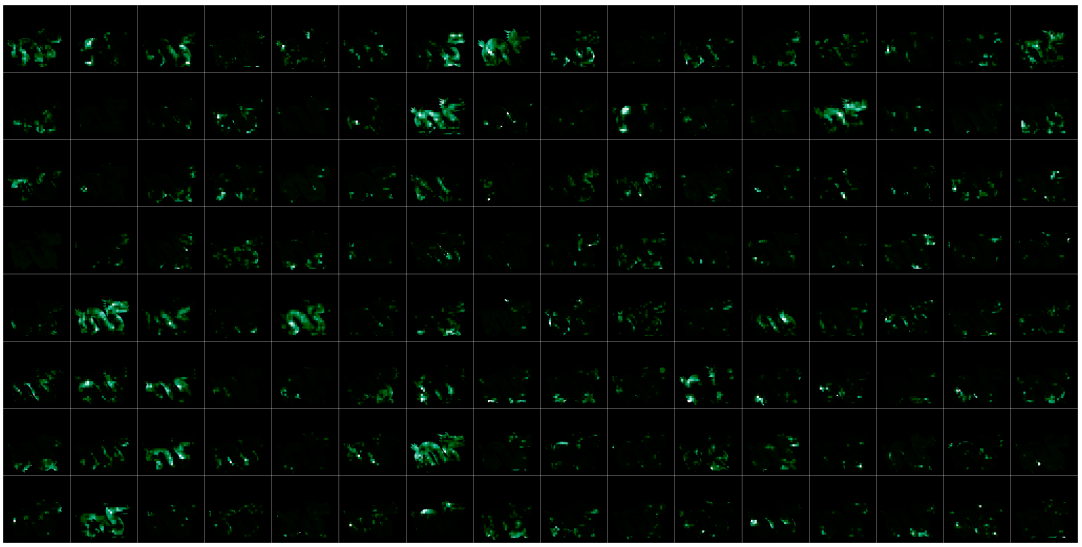


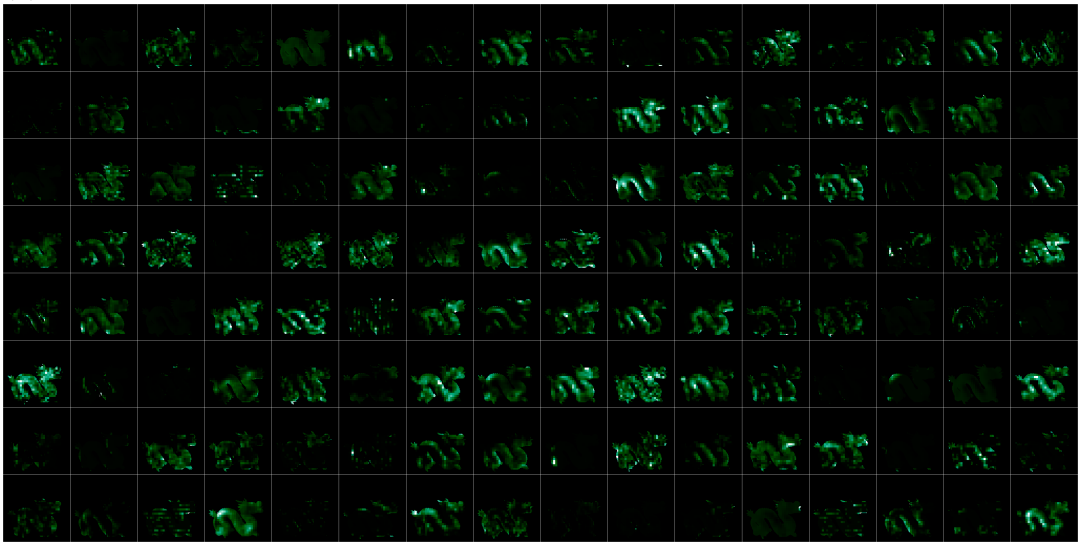
Figure B.2: Trip-Net evaluation (128×128). From top to bottom: Baoshanlu, Washington statue, Bus.

Appendix C

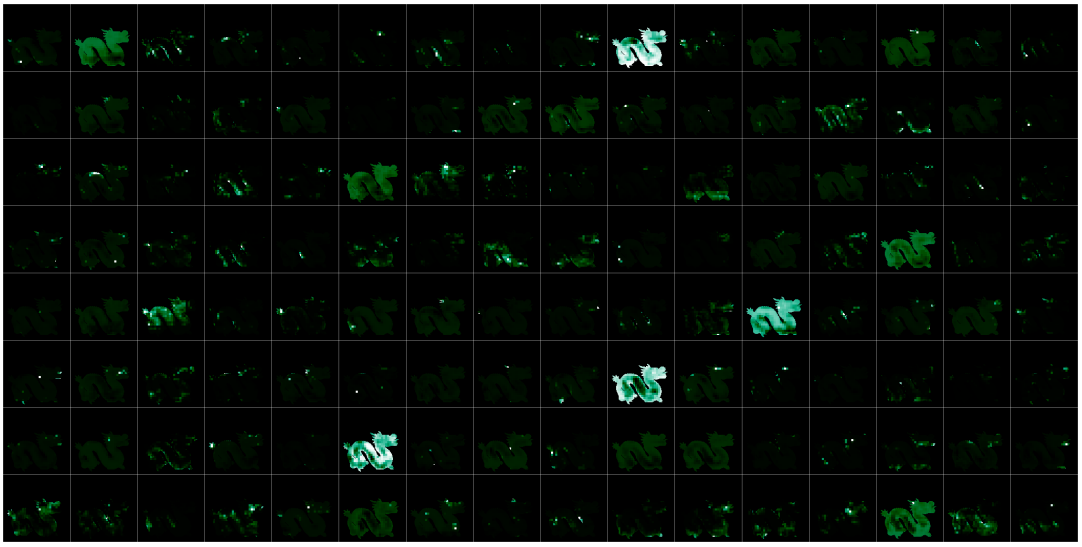
Feature Maps Visualization



(a) CNN Feature Maps



(b) NOC Feature Maps



(c) GCNN Feature Maps

Figure C.1: All 128 Feature maps of model GCNN, NOC, CNN at the first up-sampling. (Resolution 32×32)

List of Figures

List of Tables

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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