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

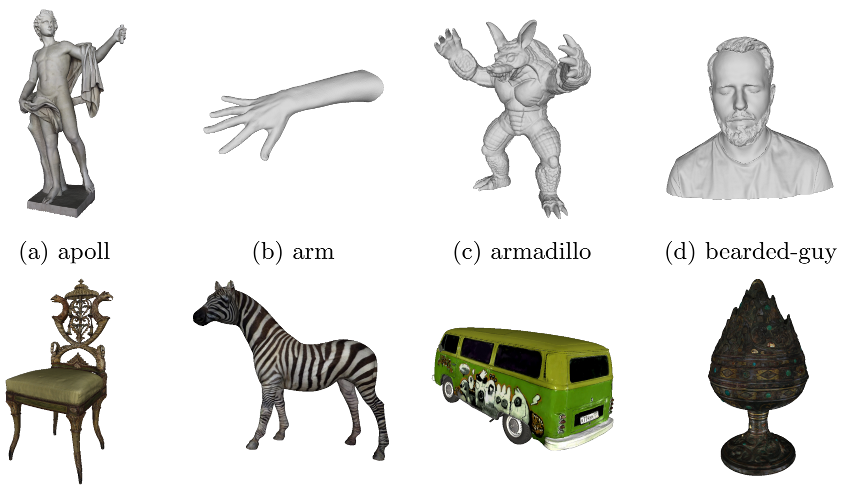
In order to acquire these data, we can use a light scanner to scan the objects in laboratory. However, the scanner is failed to scan dark, shinny and transparent region, which leaves a plenty of missing pixels and holes in the depth map. Thus the corresponding surface points are incomplete for the object.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

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

1 Resource

The object model we used are searched from internet. **data1**, **data2**, **data3** and **data4** published a set of point cloud dataset for computer vision research. These point clouds are scanned from real objects using high resolution scanners like Cyberware 3030 MS+ and calibrated with post processing. Each objects has been scanned for hundreds of times with an exhaustive completion for the origin objects, which is up to millions points. (**data1**). The dense point clouds makes the normal inference task trivial since the neighbor based method performs good enough for these kind of task. Some of the point cloud even equipped with pre-computed normal map based on more advanced methods. They all provides the accurate ground-truth for the supervised learning method.

The *synthetic50-5* is a datset we generated in this thesis with 50 point clouds as training set and 5 point clouds as test set. When we create the dataset, we tried to make the object types as many as possible to give a robust and wide range covered training scenarios. The category of the models include figure, animal, statue, toy, furniture, antique, car model, which include the smooth surface model like *arm, bus, bunny, cat, zebra* and also the model with highly elaborate details, like *Washington, Car-Engine, Armadillo*. The dataset is created base on the work of this thesis and using for normal inference task. Figure **??** gives the illustrations of some objects. Appendix **??** gives a full version of dataset models.



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

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

Figure 1: Point clouds scanned by high resolution scanners

Besides, some of the objects also has its own texture, as shown in the second row of Figure **??**. This is specially prepared for illuminated approach since they will take the image as account to evaluate the surface normal. The object surface gives a different reflectence with different surface texture, which further impact the model performance.

2 Synthesize Scenes using Unity

In order to simulate the the data collection scenario close to the real life in the laboratory as much as possible. We placed a flat cylinder called *stage* in the center in the 3D space as a platform to place objects. The stage is fixed in a determined position, which is not moved when the scene is captured. A directional light with RGB color FFF4D6 🟨 placed in a top-view direction around 25 meters away to provide as ambient light, which also has a fixed position. A RGB-D camera is placed in a top-view direction around 10m away from the stage. The camera is the sensor to capture the depth map and the grayscale image, which is random moved after a new scene. The moving range of the camera has 0.1m in both directions of X, Y, and Z axis and 1 degree random changed Euler angle. A point light placed around 5 meters away as the illumination light, which is also randomly moved after a new scene. The moving range is 0.3m in both directions of X, Y, and Z axis and 1 degree random in Euler angle.

During the data collection, the object is randomly selected and placed on the stage. The object has a random upto 30 degrees rotation align *roll*-axis, 30-degrees rotation align *pitch*-axis and 180-degrees align *yaw*-axis. Thus the camera has the opportunity to capture every direction of the object and consequently abundant the dataset. The layout in the Unity game engine is shown in Figure 2. We generate 3000 scenes with resolution $128 \times 128 \times 3$ and 5000 scenes with resolution $512 \times 512 \times 3$.

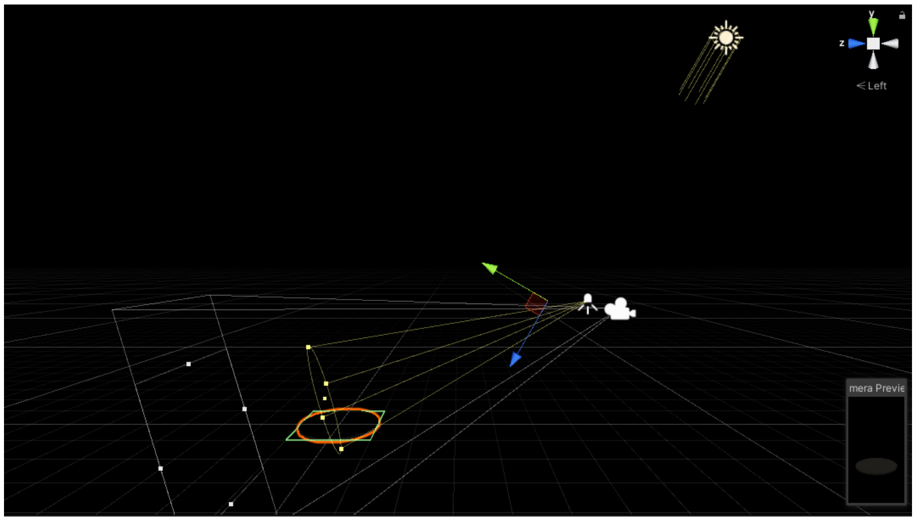


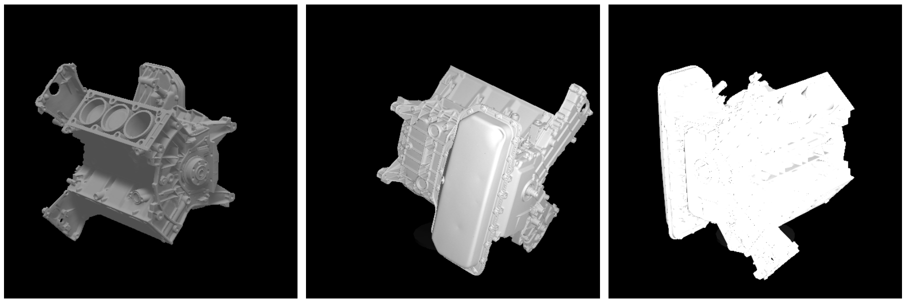
Figure 2: The layout of synthetic scenes generation in Unity.

The main advantage using generated scenes is the availability of complete information. The depth map can be captured in a loss-free way. The corresponding normal map can also be safely considered as ground truth. And the scale of the dataset is easy to control. Table 1 gives more dataset information.

One important detail that we must care about is the camera exposure. Or we have to control the light radiance on the object surface. As shown in figure 3, too less exposure on the object neutralizes the illumination information, which only left ambient light effect. Too much exposure makes the surface texture hard to recognize. When we did the experiments, we found that a suitable light setting is essential to the illuminated based approach. A too much or too less light effects won't improve the illuminated based approach.

Table 1: The information saved for each scene in “synthetic50-5”.

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



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

Figure 3: Different exposure to the objects.

3 Data preprocessing

The raw data captured from Unity required further preprocessing before feed them into the training models. A depth map is captured by a depth camera in Unity, which is a 1 channel image that contains the information relating to the distance of the surfaces of the scene objects from a viewpoint. It can be saved as a 16-bit grayscale image, i.e. each pixel in range $0 - 65535$.

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

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

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

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

$$n_z = 1 - \frac{B}{255} * 2$$

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

The camera intrinsic and extrinsic matrix helps point cloud calculation.

3.1 Convert to Point Cloud

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

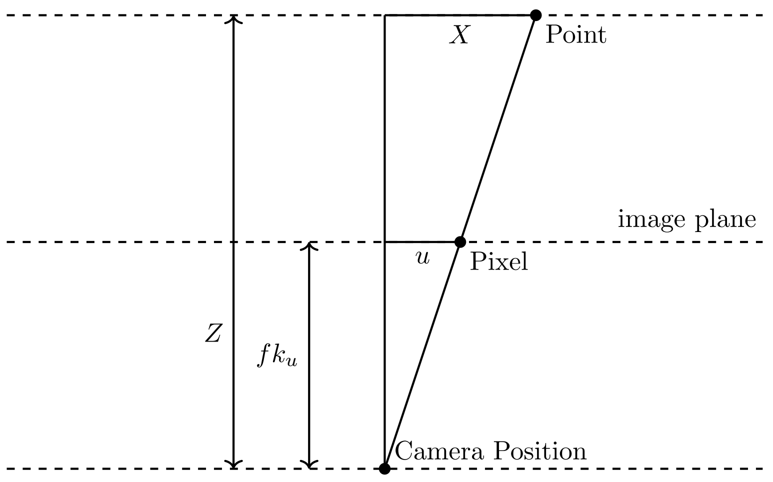


Figure 4: Convert depth to point in camera coordinate system

Consider a 3-dimensional Euclidean space. Use z axis denotes the depth. The x and y axis perpendicular with each other. For a pixel (u, v) on depth map, its depth $D(u, v)$ is the Z component of the corresponding point $P_C = (X, Y, Z)$ with respect to camera coordinate system. The X and Y can be calculated based on the triangle similarity

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

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

$$P_W = P_C R + t$$

3.2 Point Cloud Normalization

The sizes of each training object are various, whereas it should be as an invariant value for the training model. Thus the normalization is required before feed training objects into the models. The data fluctuation in each axis is shown in Figure **??**. Table 2 gives a quantitative measurement of corresponding average values.

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

$$X_n = \frac{X - \min(X)}{s}$$

$$Y_n = \frac{Y - \min(Y)}{s}$$

$$Z_n = \frac{Z - \min(Z)}{s}$$

where s is a scale factor,

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

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

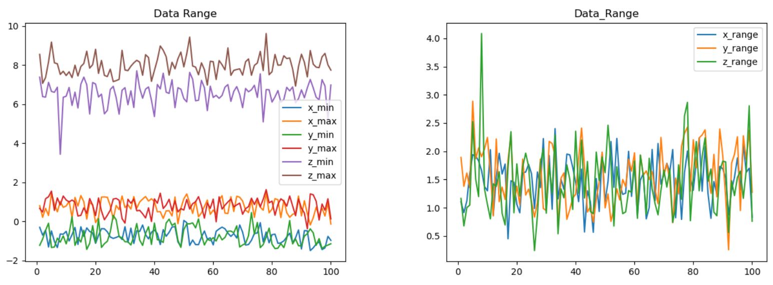


Figure 5: Left: Extreme value in 3 axis; Right: Vertex range in 3 axis

Table 2: The fluctuation of extreme values and their ranges in 100 random training items.

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

3.3 Noise

The raw depth maps captured by light scanner usually have missing pixels. In order to fit well with real-data, the synthetic data has been added uniform distributed noise, which is randomly remove the valid pixels in the maps. The simulated noise is distributed evenly around the whole map with a specific noise intensity. A parameter μ is used to control the intensity of noise, it denotes the μ-percent pixel dropoff. For example, $\mu - 10$ removes 10% pixels randomly. For each scene, the noising operation based on a random μ in a range $[0, 50]$ is used. Some scenes have more missing pixels and some have less. The random noise intensity also enables the model to learn scenarios not only with noise, but also with minor noise or even without noise. Figure 6 shows the noise effect with different μ.

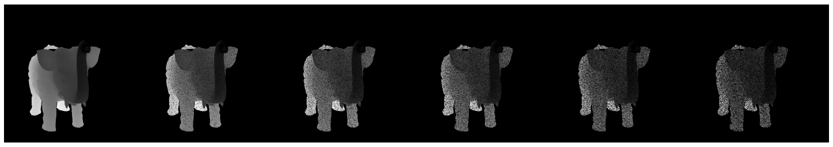


Figure 6: Noise-intensity on $\mu - 0, \mu - 10, \mu - 20, \mu - 30, \mu - 40, \mu - 50,$. Object Name: elephant-zun-lid.

3.4 Fit to PyTorch

In order to saving the training time, the dataset is compressed in PyTorch format. The structure of a single item is shown in Table 3.

Table 3: The structure of a single tensor in the dataset.

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