Learned 3D Representations: Point Cloud, Depth Maps and Holograms

LDZX7[1]

MSc Data Science and Machine Learning

Supervisor: Kaan Akşit

Submission date: September 2021

Abstract

Three-dimensional data is widely used in almost every innovative technology. However, it is questionable whether the existing three-dimensional data representation techniques are ready for future demands. Therefore, we first survey the state-of-the-art compression methods for the conventional three-dimensional data representations: point cloud and polygon mesh, because data transmission is a necessary procedure to facilitate the communication and application of three-dimensional data.

Furthermore, this thesis investigates the relationship between explicit and implicit three-dimensional data representations, including point cloud, polygon mesh, and signed distance function. In particular, we explore the translation methods between different three-dimensional data representations by exploiting their characteristics in terms of both numerical and deep learning-based algorithms and comparing the performance concerning translation accuracy and speed. More importantly, we explore holograms outside of their original intent and consider holograms as one of the three-dimensional data representations, and we propose a symmetric pipeline for encoding and reconstruction process between the point cloud and digital hologram.

Acknowledgements

I would like to thank my supervisor Kaan Akşit for the generous time and support, without whom this project would never have gone very far.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

From stereoscopic film to the near-eye display, from AlexNet [1] to PointNet [2], developing technical solutions with the help of three-dimensional data has been a hot topic in many fields for decades. And the related technologies, such as Augmented Reality (AR), Automated Driving System (ADS), and 3D printing, have raised large-scale public interest and market demand. Hence, it is clear that the demand for three-dimensional data representation and transmission will only rise in the future.

Specifically, we focus on the two central communities that widely use three-dimensional data: computer graphics and computer vision. The combination of motion capture and 3D modelling based on 3D meshes has taken the movie and game industry to a new era; also, based on the eye-tracking or visual computing, the holographic display screens and the near-eye display can give the viewer adaptive 3D perception. In short, audiences are no longer satisfied with the planar 2D display, which encourages the inventions of many 3D display technologies. Therefore, as the fundamental for these developments, the 3D data representations are supposed to be discussed as the prerequisite. Besides, to conquer the increasingly complicated 3D scenes and enable real-time performance, few new 3D data representations are introduced, for instance: the signed distance function. Since their introduction has reinforced the display performance remarkably, it is evident that choosing the suitable 3D representation and utilizing its characteristic is also a big part of an excellent algorithm. Furthermore, the application of 3D data, especially the point cloud, also takes an important place in the field of computer vision. In the beginning, due to the advancement of computing ability and emergency of large high-quality image datasets, the advent

of deep learning boosts the precision of computer vision tasks on a large scale and facilitates many technologies. The most well-known one will be autonomous driving. In particular, the current ADS are mainly based on the monocular cameras and 2D images. Although the performance has been satisfying, there are still unfortunate accidents caused by the current ADS. However, due to some natural defects from the cameras, such as motion blur and lack of direct depth information, the camera cannot be completely trusted and the investigation on 3D data usage in ADS is urgent for the safety considerations. Based on the same intuition, plenty of great algorithms [2, 4] are proposed to process the 3D input such as point clouds and learn the three-dimensional geometry, which have been showing exceptional results.

It is not surprising to see that the 3D data is in every futuristic technology because the two-dimensional representations fail to deliver the needs of current-day applications. However, it is still questionable whether the existing three-dimensional data representation techniques are ready for future demands or whether they are the most optimal form. From the aforementioned fields, it can be concluded that the main problem is how to represent an object or a scene in 3D space with corresponding depth perception. One obvious solution is adding the depth map channel to the original image. However, there is a novel 3D data representation i.e. hologram, which is able to encode a 3D scene with the 2D format. The holograms can precisely give the depth indicators by focusing from different focal distances, which solves the lack of depth problem elegantly. A detailed discussion about holograms will be given in this article subsequently.

In addition, due to the high volume of 3D data, to fully understand and explore the potential of 3D data, there are two main aspects needed to be considered, i.e. data compression and translation. First, to enhance the individual service by building a shared environment for the whole community, fast and stable data transmission is essential. Unlike the 2D images that the corresponding compression methods are well-developed, there is no standard compression method for 3D data because of its complexity. Hence, the ongoing study of the solid compression algorithm for each specified 3D data representation is a necessary angle to explore the potential usage of 3D data representation. Moreover, to leverage the capacity of 3D data and highlight the different characteristics of 3D data representations, the translation methods are essential because they ensure that any 3D data could be applied to the most suitable algorithms according to the task. Besides, the translation methods could assist the dataset construction. Since there is not enough public 3D dataset for deep learning currently, they are even more valuable.

To cover the research interests mentioned above, our contributions are as follows:

(1): To discover the future of 3D data communication, we perform a comparison between the state-of-the-art compression algorithms for conventional 3D data representations: point cloud and 3D mesh to test their compression abilities in terms of compression rate and speed; as well as a throughout survey of the translation methods between explicit and implicit 3D data representations in terms of the translation precision and the computational overheads.

(2): For the first time, we explore holograms outside of their original intent and consider holograms as one of the 3D data representations. Then, to connect digital holograms with the conventional 3D data, we propose an efficient layer-based method to encode point cloud into the hologram.

(3): A custom hologram dataset with multiple focal distance labels, and the reconstructions from focusing towards these focal distances with binary masks denoting the in-focus areas, based on our layer-based hologram encoding method.

(4): A learning-based three-dimensional auto-focusing neural network architecture based on U-net [49]. Combining with the former layer-based method to construct a symmetric pipeline for both point-cloud-to-hologram encoding and hologram-to-point-cloud reconstruction.

In summary, we would like to explore and exploit the usage of the 3D data representations, including point cloud, 3D mesh, signed distance function and digital hologram from multiple aspects in this paper.

1.2 Layout of the Thesis

In the next chapter, we mainly discuss the common usage as well as translation and compression methods of the conventional explicit 3D data representations (point cloud and 3D mesh) and implicit 3D data representations (signed distance function). By showing the previous relative works in the second chapter, there are three aims we wish to accomplish: 1, investigating the new possibilities in the field of 3D data transmission, based on the state-of-the-art 3D data compression methods; 2, by researching the current 3D translation methods, we would like to exploit and understand the links between different 3D data representations; 3, after comparing the explicit and implicit 3D data representations, i.e. point cloud and 3D mesh and signed distance function, we look for the inspiration and relevance of holograms.

As for the third chapter, we focus on another novel 3D data representation, hologram. A layer-based method for computer-generated holography with several additional features is proposed and discussed in detail. Also, a novel hologram-to-point-cloud reconstruction pipeline is proposed to explore and bridge the gap between conventional 3D data representations and digital holograms. Also, by this chance, we would like to discover the potential of the fusion between computational optics and machine learning, especially the deep learning.

In the last chapter, we will discuss the conclusions from all the previous works and the future improvement that could be tested to further benefit the results of our proposed methods.

Chapter 2

Background

In this chapter, we review the state-of-the-art (in terms of compressing rate, speed, and convenience) compression methods of two conventional 3D data representations: point clouds and 3D meshes. As well as performing and comparing the translation algorithms between the point cloud and 3D mesh. Next, we will give the experimental results in terms of a comparison table and analyze them in detail to show the differences between the methods discussed in the preceding sections and the reasons behind them, including both compression and translation algorithms. In the end, we focus on an implicit 3D data representation that has drawn attention in recent years, the signed distance function (SDF) [5]. For the purpose of seeking inspirations for the next chapter, i.e. analyzing the other abstract data form: holograms, we survey translation methods between SDFs and point cloud or 3D mesh.

2.1 Overview

Recent years have witnessed the growth of three-dimensional (3D) data-based applications in many fields, such as autonomous driving [6] and augmented reality [7].

Thanks to the latest advances in 3D sensing technologies, a growing number of 3D data acquisition devices have become affordable, and the amount of available 3D datasets has also increased rapidly. Due to the availability of 3D data and breakthroughs in 2D deep learning architecture, especially the convolutional neural networks, investigating and developing the combination of deep learning architectures and 3D data input has become valuable and vital. Since the 3D data can provide more compact and fine-grained information about the three-dimensional geometry of objects or environment than 2D images,

deep learning methods based on 3D input have great potential to outperform the current image-based computer vision algorithms practically in many tasks such as classification, detection, and segmentation.

Besides, in the applications of computer graphics and display, the demand for 3D data is enormous. As proposed in [9], the future display equipment is not restricted to 2D images, but the holographic display for 3D scenes, which means a large amount of 3D data is needed as the necessary fundamental. Not to mention the movie and game production, the rise of these industries is mainly due to the rapid development of 3D modelling, rendering and shading technologies.

There are two main conventional 3D data inputs: point cloud and 3D mesh. In particular, firstly, the point cloud as the most fundamental 3D geometry representation which consists of the coordinates of a point set, it can be recorded directly by the sensors, e.g. LiDAR. However, the point clouds are unordered and sparse in the three-dimensional space. Hence, current technologies such as convolutional neural networks need additional operations before processing the point clouds, such as voxelization. In contrast, the 3D meshes define the 3D object by a collection of structured polygons and store the 3D geometry and connectivity separately. The additional topological information gives mesh huge advantages in rendering, although a high-resolution mesh needs plenty of time to model by specific software. Based on these two representations, there are new 3D representations introduced in the last few years, e.g. signed distance function, to cover the defects of conventional 3D data representations and target to unique aims.

However, 3D data has its drawbacks compared to 2D data. First, although the cost of obtaining 3D data has reduced significantly because of the improvement of 3D scanning technologies, the price of a 3D scanner is still much higher than an ordinary camera. And this will make the 3D technologies hard to spread for the public uses. More importantly, the 3D data storage size is vast. So, without a reliable compression method, it will be a significant burden for the current data transmission bandwidth to transmit a high-precision dynamic 3D model sequence, which may result in the breakdown of real-time applications of most 3D technologies, as well as the failure of information sharing between multiple users.

To address this problem, we believe that utilizing the powerful compression algorithms can further benefit the performance of subsequent algorithms by enabling efficient and effective data communication and data augmentation to, for instance: building a shared map of the environment to enhance the safety of autonomous driving.

Also, compared to the 2D data, the information contained inside the 3D data is more complex, so there are multiple types of 3D data representations with various characteristics and are applicable for different algorithms. Then, in order to fit the 3D data into the most suitable algorithm regarding its mission, the translation algorithms are required. Besides, a high-precision translation method could facilitate the same 3D dataset for different purposes.

Hence, in the following subsections, we will review some latest outstanding point cloud and 3D mesh compression methods that cleverly leverage the characteristic of each representation, as well as the translation algorithms between the point cloud and 3D mesh. To end with, we will add the discussion about another unique 3D data representation: signed distance function (SDF), including the methods of converting SDF-to-point-cloud and SDF-to-mesh [8], and vice versa.

2.2 Point Cloud Compression

Point clouds are becoming increasingly preferable 3D data representation in many advanced applications. But, unlike images, point clouds are sparse and unordered in three-dimensional space, and due to these properties, it is hard to transfer the successful ideas of 2D compression to the point cloud compression directly. Hence, there are three leading families of 3D point cloud compression approaches that have been proposed [10]. The first one is 1D traversal compression, it finds the neighboring relation of points in the cloud and encodes the connectivity information into trees, then a 1D prediction-adapted signal can be constructed via various traversal order so that the original explicit coordinate information is compressed.

Although this tree-based method is simple and easy to implement, only a limited compression rate can be reached and limited bounded space to improve. Therefore, the Moving Picture Expert Group-3D Graphic group (MPEG-3DG) has been encouraging the other two types of point cloud compression (PCC): Video-based PCC (V-PCC) and Geometry-PCC (G-PCC) [11]. Thanks to the existing high-efficiency image or video compression methods, the idea of V-PCC is projecting or mapping the point cloud into 2D planar grids, then the 2D compression methods can be applied directly. The G-PCC compression directly exploits the 3D geometrical correlation inside the point cloud with the help of quantization.

Furthermore, the combination of deep learning and PCC has become popular recently.

For instance, [12] proposed a deep learning-based lossless compression method for point cloud geometry. [12] first partitions the point cloud into the 3D grid to make a general convolutional neural network applicable and saves the voxelized point cloud geometry into an n-level octree adaptively to reduce the sparsity by a novel rate-optimized multi-resolution splitting algorithm. In other words, the algorithm above first detects and splits the non-empty voxel only to cut the unnecessary computation and data storage amount. Then, the authors proposed a novel architecture to encode and decode the input point cloud called VoxelDNN, the points are passed through VoxelDNN by sequence, and the trained VoxelDNN predicts the probability distribution of a specific voxel is being occupied based on all the information of previous voxels by a masked 3D convolutional neural network. This work is inspirational because it increases the efficiency of PCC by exploiting the sparsity of point clouds and succeeds in mixing deep learning and PCC.

In order to further reduce the redundancy in volumetric methods, several point-based approaches have been proposed, such as [13]. Due to the well-developed sparse convolution in three-dimension space, these point-based methods have achieved impressive results. Particularly, we would like to highlight another learning-based method proposed by [14] (MPCGC) because this method achieves both outstanding performance and high efficiency.

[14] proposed a method that compressing the point cloud geometry and its feature attributes separately, but they are both compressed and decompressed by the sparse convolutions powered by *Minkowski Engine* [15]. To be specific, for encoding the given point cloud, MPCGC extracts the key points progressively by repeating a similar network structure, which consists of two 3D sparse convolution layers with the rectified linear unit (ReLU) as activation functions, as well as three additional stacked Inception-Residual Networks (IRNs) [16] units to enhance the feature extraction (as shown in the Figure 2.1).

After several times downscale sampling the input point cloud, the geometry entries of the critical points are passed through an octree encoder. Meanwhile, a uniform quantization is applied to the feature entries, then the quantized features are compressed into an arithmetic encoder using a probabilistic model, as well as an extra entropy model for recovering the information loss caused by the probabilistic model. Note that after the aforementioned operations, the input point cloud is compressed into two binary strings, one for the geometry information, the other for the features information. As for the decoder, MPCGC mirrors the encoding process to reconstruct the point cloud gradually by re-sampling. The only difference of the network structure is that the decoder additionally classifies the occupied voxel by performing binary classification according to the voxel-being-occupied probability

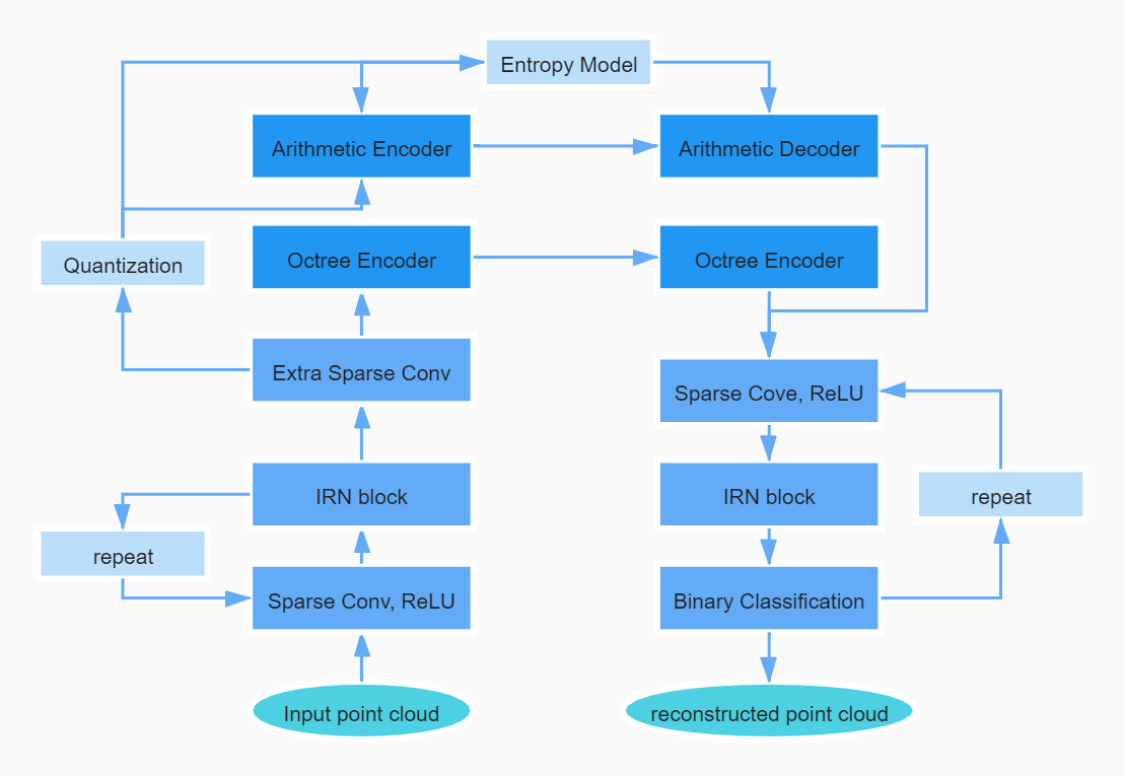

Figure 2.1: The network architecture of MPCGC.

generated by employing sparse convolutional layers. The classification aims to enable the progressive refinement of the reconstruced point cloud resolution, so the reconstruction quality is enhanced hierarchically, and the complexity can be reduced. Note that the *repeat* blocks in the Figure 2.1 mean repeating the corresponding operations several times but with different input and output dimensions. Besides, the empirical experiments suggest that the optimal number of repeat iterations should be three.

Overall, [14] proposed a handy PCC algorithm, which has impressive performance in both compression rate and speed by utilizing various methods to overcome the sparsity nature of point cloud.

2.3 3D Mesh Compression

Unlike the point cloud, the 3D mesh has been the most popular 3D representation for years, and it is widely used in many 3D graphics applications such as animation, film production, and game development. Also, the 3D mesh contains both geometry and connec-

tivity information, thus the compression could be improved by exploiting the topological information about the encoded 3D shape. Therefore, the compression technologies have been mature for 3D mesh, especially the triangle mesh.

In this case, we choose several well-developed 3D mesh compression software packages as our candidates, which are basic glTF file compression [19], Open3DGC (Open-3D graphic compression) [17] and Draco [20]. The glTF file is compatible with multi-platform and multi-language, but its compression rate is unsatisfactory. In the end, Google's Draco performs the best among the compared in terms of compression rate and speed (as shown in the Table 2.1). Additionally, the Open3DGC uses a lossy compression algorithm which includes quantization [18], but the Draco allows lossless compression by providing the users with a loss rate parameter to balance the tradeoff between compressing loss and compressing rate.

	3D model	Bunny	Dancer
	original size	2318 KB	134006 KB
Draco	compressed size	116 KB	1627 KB
	encode time	0.15 s	3.7 s
	decode time	0.03 s	0.53 s
Open3DGC	compressed size	94 KB	2524 KB
	encode time	0.7 s	13.6 s
	decode time	0.2 s	1.9 s

Table 2.1: Comparison table for different 3D mesh compression methods.

Although the Draco library is open source, there are no official documents available online explaining the inner mechanism of the Draco package. Then, by inspecting the source codes and the references, we find the EdgeBreaker algorithm [22] plays an important role that accelerates the performance of Draco.

EdgeBreaker is firstly proposed in [21], as its name suggests, the EdgeBreaker offers a an efficient and straightforward compress/decompress method for connectivity by 'breaking' the edges of the mesh by sequence. Since the vertex locations are independent of connectivity, the geometry information can be compressed through a predictive scheme based on the connectivity information, and combining with entropy encoding can further reduce the data occupancy and increase the decoding accuracy in practice, especially for large meshes.

In other words, for encoding the given 3D mesh, the EdgeBreaker algorithm traverses one triangle to its neighboring triangle deterministically and spirally and assigns one de-

scriptor from set {C, L, F, R, S} for each visited triangle to form a sequence string called clers. One descriptor describes the surrounding structure of the current triangle with respect to the common untraveled vertex of the current triangle and the next triangle. In other words, the state of the current triangle is recorded in the descriptor. The geometry situations corresponding to different descriptors are shown in the Figure 2.2, where v is the current vertex/corner, X is the current triangle, the thicker edge is the current active gate, the red arrow points to the direction of the next triangle. Note that the algorithm traverses to the right triangle first and then to the left in the S case.

In addition, trivial binary codes {C=0, L=110, F=111, R=101, S=100} is used to store the clers in binary format, and the Huffman coding could be applied to reduce the file size further. Then, as for decoding, EdgeBreaker takes the binary clers and reconstructs the target mesh by appending the new triangle to the previous triangle according to the clers string. In the end, EdgeBreaker confirms the vertex locations, i.e. coordinates by the entropy decoding and zips every part of the 3D mesh together; hence the original mesh input is reconstructed successfully.

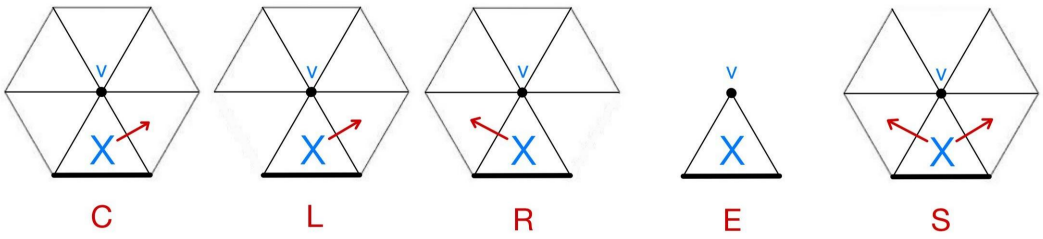

Figure 2.2: Visualization of the geometry condition corresponding to each descriptor of EdgeBreaker.

Furthermore, in [22], the authors proposed an advanced version of EdgeBreaker that uses a new data structure, called Corner-Table, to encode the 3D triangle mesh that can allow the EdgeBreaker implementation even simpler and faster. The Corner-Table is simply two arrays of integers, which considers the triangle as primitive and represents the connectivity of a triangle mesh by indexing. In other words, the connectivity array is separated into two parts, V and O, which store the integer index of vertices and their opposite corners. Also, during the compression and decompression stage, two extra temporary arrays are generated, M and U, which indicate whether each vertex or triangle is traversed to guide the EdgeBreaker agent to the untraveled triangles. Then, the EdgeBreaker algorithm is adapted for this particular data structure, while the main idea is still the same. More

surprisingly, the length of the total codes is no more than two pages of A4 paper. The simplicity makes EdgeBreaker more suitable to fit in a software package.

2.4 Data Translation

In this section, we test various translation approaches of transforming point cloud to mesh or vice versa. The purpose of this section is to discover the computational overheads of 3D data translation; thus, the results can be used while considering the efficiency of applying data translation as data augmentation.

The method of mesh-to-point-cloud translation is straightforward, which is sampling points from each face of the 3D mesh. The sampling process can be extended to be progressive or adaptively uniform to ensure the level of details (LOD) are preserved well in the resulted point cloud. Since the method is straightforward, there is little literature related to this topic, and the practical implementations are contained in most 3D software packages.

2.4.1 Ball Pivoting Algorithm

As for the point-cloud-to-mesh translation, there are two main approaches. The first is the Ball Pivoting algorithm (BPA) [24]. The main idea of BPA is using an imaginary tiny ball rolling along the outer surface of the input point cloud. Then, BPA forms the new triangle by linking the points which touch the imaginary ball simultaneously and stop the ball from falling into the surface. This process continues until all the points are traversed, and the target 3D mesh is complete.

BPA is simple to understand, but careful hyper-parameter settings, especially the radius of the ball, must be tested before actual application. Because using inappropriate hyper-parameters may lead to multiple potential problems, for instance, the ball may miss the narrow 'valley' of the 3D geometry such that the details of the surface would be lost. Hence, the BPA implement requires high standards of the input point cloud, which makes BPA hardly an optimal choice in practice.

2.4.2 Poisson Reconstruction

Another handy method is the Poisson reconstruction [25]. Poisson reconstruction receives a set of oriented points, i.e. a cloud of points with their normal vectors, then outputting the reconstructed 3D mesh. The main idea is using the normal vectors from

the input samples to compute the precise 3D piece-wise indicator function to represent the 3D geometry. The gradient field is defined and approximated firstly in order to avoid the infinite boundary value computed by the indicator function, [25] proposed a lemma that allows the gradient field can be defined via a smoothed indicator function. Then, although the geometry information cannot be known from the unordered point cloud, the surface integral can be approximated by summation over each discrete point patch. Next, since the continuous vector field cannot be integrated directly, the least-squares solution is derived by solving the Poisson equation, which is formed by applying the divergence operator to both sides of the equation $\nabla X = V$, where X is the target equation and V is the calculated vector field. Finally, the isosurface of the indicator function, in the form of a 3D mesh, is calculated by the Marching Cubes Algorithm [26] (more details about the Marching Cubes Algorithm are in the following subsection **1.2.4 signed distance function**). Additionally, the octree is employed for the Poisson Reconstruction to cut the redundancy in the 3D grid.

There are various advantages of Poisson reconstruction: 1. the algorithm could produce smooth and watertight surfaces; 2. the resulted 3D mesh is much more robust to noise in the data than BPA. Also, in the practical implementations, Poisson reconstruction only has one main parameter *depth* which controls the level of details for the output mesh, resulting in a user-friendly API (Application Programming Interface). Disadvantages of Poisson reconstruction is also apparent: the reconstruction needs precise normal vectors which cannot be measured directly or calculated precisely. This defect makes the Poisson Reconstruction even slower given its heavy computation.

Compared to the BPA, Poisson reconstruction can capture a higher level of details, as well as the quality of reconstructed mesh via Poisson reconstruction does not highly depend on the hyper-parameter settings. Although Poisson reconstruction is handier, a complete understanding of the mechanism behind Poisson reconstruction requires massive mathematical and topological knowledge background.

2.5 Experiments

In this section, we present our experimental results of two leading compression and translation methods concerning two main 3D data representations, point cloud and 3D mesh, in terms of a detailed comparison table.

To maintain the fairness of the experiment, two compression methods are applied to

the same 3D model, which is the first frame of the 3D dancer model sequence from one of the MPEG authorised point cloud compression datasets [23]. The original file is provided in point cloud format, which contains 2592758 points, to transform the point cloud into a 3D mesh with similar resolution, the Poisson reconstruction method [25] with considerable depth $depth = 10$ is chosen for the equivalent accuracy and smoothness. The translated mesh contains 1557063 points and 467189 triangles. Since the main aim of the experiments is to show 3D geometry compression, feature attributes are excluded from the original file. Note that the experiments are performed through a 2.5GHz Intel-i5 CPU without using CUDA GPU acceleration. All the numerical entries have been averaged to avoid randomness.

	Point Cloud	3D Mesh
compression rate	$\frac{93.1Kb}{145.3Mb} = 0.525\%$	$\frac{1.62}{134}Mb = 1.20\%$
	0.0240 bits per point	-
code storage size	3.1 MB	$\approx$ 2.8 MB
encode time	34.8 s	3.7 s
decode time	35.3 s	0.53 s
encode CPU usage rate	73% (average)	62% (peak)
decode CPU usage rate	74% (average)	68% (peak)
reconstruction accuracy	lossless	lossless
translation time	17.7 s (to 3D mesh)	3.7 s (to point cloud)
translation complexity	O(size of reconstructed mesh)	O(target number of points)

Table 2.2: Comparison table of point cloud and 3D mesh compression and translation methods.

Firstly, we compare the compression rates of each compression method. As shown in the Table 2.2, the compressed point cloud model is only 0.525% of its original size and achieves around 0.0240 bits per point (bpp) which is very high compared to its competitors in the field of point cloud compression. Besides, the 3D mesh version is compressed to have 1.2% of its original size by Draco, which also shows an impressive result. Although the compression rate of the point cloud is much lower than 3D mesh, the computational burden and energy consumption of point cloud compression is much higher than mesh compression. Because learning-based point cloud compression needs to be appropriately trained until being used, in contrast, the Draco package can be used as a plug-in application since the EdgeBreaker algorithm can compress the target mesh directly.

Besides, the size of the whole Draco package is 75.9 MB (unzipped); however, after pruning the unnecessary codes, e.g. API support for other 3D modelling software, the size

can be significantly reduced to around 2.8 MB (since Draco is built on C++, in order to use Draco via Python, a Python wrapper for Draco will cost extra 473 KB storage amount). In the meantime, a trained network for point cloud compression occupies 3.1 MB storage space.

As for the compressing and decompressing speed, first, we can see that the decoding time for a point cloud is similar to its encoding time, because the decoder network mirrors the encoder network, and the additional classification operations in the decoder do not spend much extra time, only 2.36 seconds totally in our implement. On the contrary, the reconstruction time for compressed 3D mesh is much less than its compressing time. As described in [22], unlike the encoding process which sorts each vertex of the mesh by multiple *if/else statements* to form the clers, the decoding algorithm only needs to transform the corner-table to rebuild the geometry and connectivity arrays asynchronously according to the encoded clers, so the decompressing time is significantly less than the compressing time of the EdgeBreaker.

Although the encode/decode time for 3D mesh compression outperforms the point cloud compression via CPU implementation, note that the learning-based point cloud compression is based on the deep learning platform Pytorch [59] so that the encode and decode speed of point cloud compression can be scaled down significantly with the help of GPU acceleration. For the original hardware settings in [14], compressing and decompressing the same object only takes 0.334 and 1.368 seconds with an Intel Core i7-8700K CPU and an Nvidia GeForce GTX 1070 GPU. However, the computing cost of the neural network would be much more than the typical compression methods in general.

In conclusion, the purpose of the throughout comparison is to give us the deep and explicit instruction and understanding of the practicability of each method. Furthermore, we compare the complexity of each 3D data translation method mentioned in section **1.2.3 data translation**. Firstly, the significant difference between 3.7 and 17.7 seconds is intuitively reasonable because the Poisson reconstruction [25] needs much more calculation and operation than the sampling. But, if the translation quality score is required to be very large, i.e. a relatively high level of details, the computational complexity is still linear with the size.

In addition, since all the experiments above are done with Python, the primary purposes of these models are for performance evaluation and to show the practicability and feasibility of the proposed methods, the encode and decode time shown in the table can only provide a general sense about the complexity of each method. Besides, to help speed

up the compression process mentioned above, a more hardware-friendly programming language could be used, for example C++.

The two compression methods can both get lossless reconstruction accuracy, but a few parameters can be adjusted to balance the tradeoff between the accuracy and compression rate or speed. Here, to maintain the fairness of comparison, we only compare the performances of 3D data compression methods under the lossless compression condition.

2.6 Signed Distance Function

In mathematics, the signed distance function (SDF) represents a function $f(\cdot)$ that maps the location of a point x to the distance between the point and a pre-defined implicit boundary $\partial\Omega$, where Ω is the corresponding subspace determined by the boundary. The magnitude of the result represents the distance defined by a metric, and its sign shows whether the point is inside or outside the boundary $\partial\Omega$.

$$f(x) = \begin{cases} d(x, \partial\Omega) & if x \in \Omega \\ -d(x, \partial\Omega) & if x \notin \Omega \end{cases} \tag{2.1}$$

where $x \in \mathbb{R}^3$.

Therefore, the boundary of SDF defined in three-dimensional space is capable of representing any continuous 3D shapes. Due to the flexibility of SDFs, they are applied in many fields, particularly for game production; SDFs can solve many complicated problems such as direct shadowing in an elegant way. Despite the various advantages of SDFs, high-precision SDFs can be hard to be generated directly because they are implicit shape representations that need derivation before application. Hence, we will focus on the translation methods between mainstream 3D representations and SDFs.

Firstly, given an SDF, they are several choices of transforming SDF to other conventional 3D representations, enabling the downstream usages, for instance: visualization. For SDF-to-point-cloud, sampling uniformly around the approximated zero-plane of the SDF is a common idea, the corresponding process is similar to a virtual 3D scanning. But, for SDF-to-mesh process, it is more complex than SDF-to-point-cloud, and the Marching Cubes Algorithm (MCA) [26] is introduced for this issue.

2.6.1 Marching Cubes Algorithm

As its name suggests, the SDFs cannot be visualized directly until sampling to another explicit 3D representation such as 3D mesh, so this is where the MCA is needed.

MCA is the typical method for locating the isosurface from the discrete 3D scalar space. The whole metric space is first sliced into the 3D grid; then, the algorithm processes the SDF by marching along the cubes. The vertices of each cube, i.e. 8 neighboring pixels, can be marked into two classes: the vertex is either inside or outside the boundary defined by the SDF. And this can be identified by the sign of its SDF output. However, there are $2^8 = 256$ possible situations for the vertex distribution of each cube. Since the vertex distribution remains unchanged if the cube is reversed, the number of possibilities is halved. Also, due to the rotation invariance, the number of possibilities are further reduced to only 15 (including the empty cube).

Hence, 15 index indicators for any specific cube are enough to show the intersection condition of each edge and the isosurface, and it could be confirmed by checking the distribution of signs of 8 vertices. Then, the edge information can be read from a pre-calculated table (triangle table) regarding its index and the surface-edge intersection, i.e. the locations of triangle vertices is computed through applying linear interpolation based on the densities of the neighboring edge vertices; finally, the triangles are joint together to form the resulted 3D mesh. The fused triangles will be the 3D mesh which represents the isosurface/zero-contour of the given SDF.

In addition, the normal vectors of the mesh at each vertex can be computed efficiently by taking the derivative of SDF and interpolating to the vertices. Furthermore, we can speed up MCA by incorporating the idea of octree or KD-tree [27], which reduces the redundancy in the metric space and narrows the search area. The drawback of MCA is quite apparent, as one of the iterative algorithms, the efficiency of MCA is a remaining issue that requires parallel computation or hierarchical structure.

2.6.2 DeepSDF

As for SDF generation from other explicit 3D shape representations, e.g. mesh or point cloud-to-SDF, due to the fast-paced improvement of deep neural networks (DNNs) as global function approximators, DNNs have been discovered to approximate the SDFs and achieved great success. In this case, we would like to stress one successful approach that transforms input point set to continuous SDF, called DeepSDF [28]. The most important

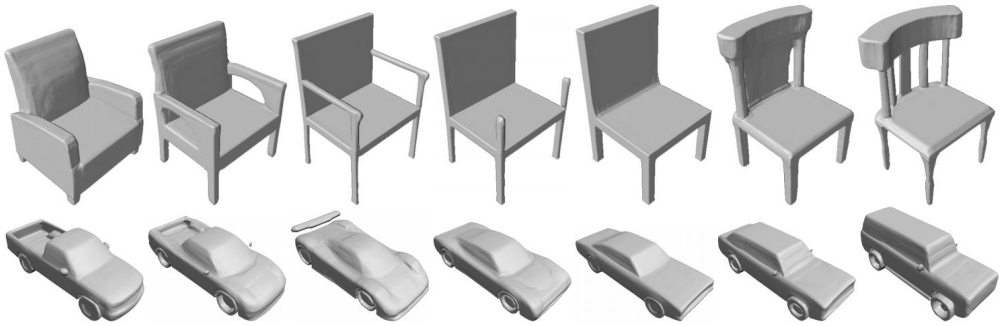

Figure 2.3: Visualization of the synthesized 3D models by DeepSDF [28]. The two classes of objects interpolated by DeepSDF corresponds two trained latent code-conditioned feed-forward decoder networks.

contribution of DeepSDF is that, rather than representing a single 3D shape, DeepSDF can model a class of 3D models by a decoder-only network architecture.

The main idea of DeepSDF is that considering the target SDF as the decision boundary represented by the DNN, where the signs are the network outputs. The implementation of modelling a single shape is relatively simple, DeepSDF trains a DNN f based on input point sets with the accurate distance and sign, such that

$$SDF(\mathbf{x}) = d \approx f_\theta(\mathbf{x}) \tag{2.2}$$

with respect to the minimization of clamped L_1 loss, where $\mathbf{x} \in \mathbb{R}^3$ is the input point location, $d \in \mathbb{R}$ is the distance to the shape boundary with particular signs, f_θ is the target DNN with its parameters θ.

In addition, in order to model a class of 3D shapes based on the auto-encoder model, [28] proposed an auto-decoder network framework. Then, the network takes one more latent vector $\mathbf{z}$ as input, which indicates the class that the point set belongs to. As for the training process, to take the latent vector $\mathbf{z}$ into consideration, the problem is formulized into a probabilistic format. And the latent shape vector $\mathbf{z}_i$ for i-th sample X_i can be computed by the Maximum-A-Posterior (MAP) estimation.

The advantages of DeepSDF are: 1. the output SDF, i.e. trained neural network, is light-weighted, which only occupies less than $10MB$ for representing each class of shape; 2. in the meantime, the resulted SDFs can support any levels of resolution for complex 3D models. Despite the aforementioned advantages, the training of DeepSDF needs heavy data preparation and augmentation because DeepSDF only receives point-based data and cannot process topological information, for example the connectivity of 3D mesh, as well

as DeepSDF assumes the shapes are in standard poses.

Moreover, NVIDIA's newest research [29] proposed the current state-of-the-art 3D reconstruction technology, which outperforms DeepSDF in both quality and speed. These works have shown the unlimited potentials of SDF as an implicit 3D data representation in rendering and displaying. In the next chapter, we will discuss another novel 3D data representation: holograms, which also promises the future of display.

Chapter 3

Holograms

In this chapter, we will discuss a novel three-dimensional (3D) representation: hologram. The optical holograms are first introduced as the background knowledge; then, we focus on the computer-generated holography in terms of 3D digital hologram simulation and reconstruction methods.

The term Holography refers to recording the light diffraction and inference pattern reflected from the object surface into a hologram so that the original 3D scene could be reconstructed from a continuous range of angles and depths. The reconstruction of a hologram needs another coherent light beam which is the same as the recording beam to shine on the recording medium; the depth and focus cues are well-defined naturally in such reconstruction, therefore giving an elegant way of solving related digital display problems. Furthermore, similar to the signed distance functions, holograms could describe 3D shapes in a continuous manner, which enables holograms to be extended to a high-resolution representation. More importantly, the corresponding depth map of the input 3D scene is encoded inside the hologram implicitly, which is given in terms of the focus cues.

Currently, holograms are widely used in many fields such as security, biosensor, data storage, etc. To break the limitations of optical holography, computer-generated holography is introduced to produce digital holograms from virtual 3D scenes by simulating light propagation and diffraction pattern with the aid of the computer.

As for hologram compression, the 2D-shaped data in holograms, especially the phase-only or amplitude-only holograms, can be passed to well-developed JPEG lossy compressing method [30] directly. But unlike the general RGB images where the neighboring values change smoothly, the hologram contains much more high-frequency information, and its neighboring entries differ rapidly; thus, the compression decoder needs assistance from an

additional tool such as a neural network to recover the high-frequency loss [31]. Therefore, compared to the compression methods for previous conventional 3D representations, i.e. point cloud and 3D mesh, the digital hologram has natural advantages in data compression. In addition, due to the same reason, the general 2D convolution can be directly used to hologram. This feature provides the massive convenience of applying the succeeded image-based convolutional neural network architectures to the digital holograms.

However, the computational limitation and display barrier prevent holograms from being used publicly, we believe that there are still many undeveloped potentials regarding digital hologram encoding and reconstruction.

3.1 Optical Holography

Holograms can be considered as '3D images' which are recorded by an optic process called holography.

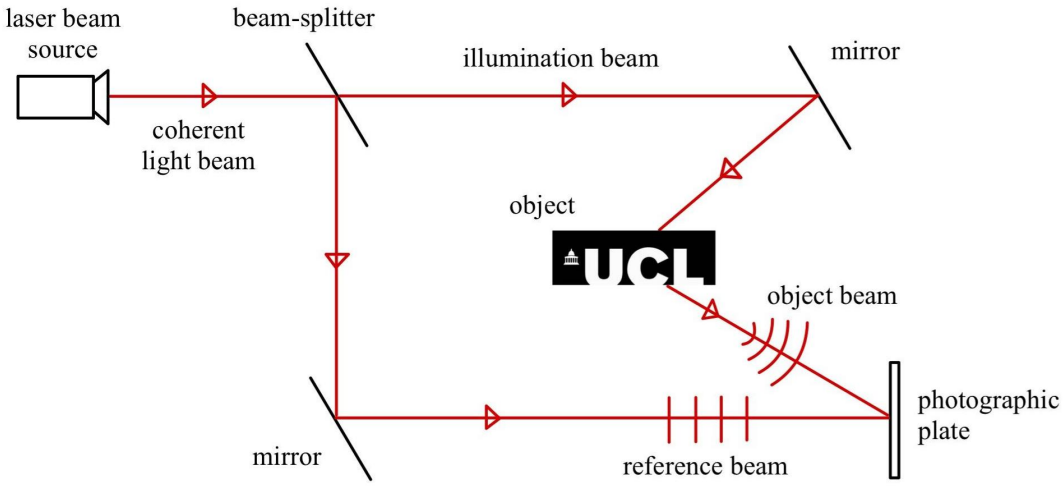

Figure 3.1: Visualization of the optical holography experiment setup.

The basic holography experiment setup is shown in Figure 3.1. As shown in the diagram, the original laser beam, i.e. a coherent light beam, is separated into two identical sub-beams by a particular lens. Then, two beams are redirected by mirrors, the illumination beam illuminates the given object to form the object beam, and the reference beam shines directly to the photographic medium for recording the hologram simultaneously. By superimposing the reference beam and the object beam, the interference pattern is saved

into the photographic medium so that the original scene is recorded three-dimensionally as a hologram, and the resulted hologram can be reconstructed by shooting the same laser beam to the recording medium subsequently.

Compared to photography, holography can record amplitude and phase information of the given scene from the inference pattern. Therefore, the reconstructed scene inside the hologram can be viewed in multiple angles, providing more accommodation cues and giving the same perception to the human brain as the original scene. On the contrary, both recording and reconstruction processes of holograms require special, i.e. coherent light beam, while the photographs can be recorded and viewed under any regular light sources.

In addition, unlike the physical imaging process in which the quality of the image could be damaged by overexposure, holography does not suffer from overexposure issues, and usually, the longer exposure time would result in better hologram quality. More importantly. the optical holography process requires an extremely stable experimental environment; slight vibration, air turbulence, or leaked light would damage the hologram significantly, therefore the environment must remain wholly motionless and dark to ensure the quality of the hologram. Due to this limitation, the hologram recording time should be controlled carefully for the tradeoff between quality loss and experimental bias.

Finally, we can find that holography is hard to be implemented physically because of its high requirements. However, thanks to the rapid development in computational optics and hardware computing power, it is now possible to simulate the complex light field propagation by the computer which realizes the potential of computer-generated holograms.

3.2 Computer-generated Holography

With the increasing demand for 3D holographic display technology in augmented reality (AR), virtual reality (VR) and simulated reality (SR), we would like to stress the great potential of holograms in the 3D digital display field by introducing the computer-generated hologram (CGH). Although the current computational limitation restricts the resolution of light field propagation simulation, we believe that the CGH technology is still worth exploring and has a great possibility to become the next focused area in the future.

Computer-generated holography has several advantages over optical holography. Firstly, the rendering of CGH only requires a computer with relatively high computing power instead of a high-standard optical experiment layout. Secondly, since the CGHs production

is virtual, CGHs can be synthesized by unreal scenes, such as the computer-aided design (CAD) models or virtual scenes.

Currently, the CGHs are generally produced by point-based methods or polygon-based methods. To be specific, for point-based methods, the 3D object or 3D scene is sampled into plenty of points, then each of the points is considered a source of the spherical wave; therefore, the hologram could be fused by superimposing all the spherical light waves from every point. The polygon-based methods are built based on the principle of point-based methods, but polygon-based methods consider the polygons as primitives to reduce the calculation time. And the ray casting algorithm is employed to simulate the light projecting from each faces for the polygon model. Note that the input 3D data representations of the methods above are point cloud and 3D mesh respectively, which are the central 3D data representations discussed in the previous chapter.

Although the computational time is reduced in the polygon-based methods, the total runtime can still be unaffordable for real-time applications or high-resolution CGHs. Hence, we aim to discover an efficient and accurate algorithm to translate the conventional 3D data, i.e. point cloud, to the digital hologram.

3.2.1 Layer-based Hologram Translation

In this section, we propose a fast computer-generated holography routine. Based on the previous works in [34, 35, 36, 37], in short, our layer-based method separates the input 3D model into several slices and simulate the light propagation individually, finally superimposing the sub-holograms together by addition to generate the overall digital hologram. However, to differ from the previous works, our implementation includes several additional features, e.g. depth-fused 3D (details in the following subsections) to improve the quality of generated digital hologram. Also, our implementation avoids using any loops for encoding points to their corresponding planes and diffraction simulations, i.e. a well-designed vectorization is introduced to increase the efficiency.

To perform our layer-based method, first, several imaginary planes are placed parallel to the XY-axes, and the location of each plane is equally separated and set according to the pre-defined number of planes and the depth of the input 3D scene. Then, the input is sampled into point cloud format, and each point is classified to find its nearest plane; the corresponding plane index is assigned to each point. In the next step, all the points are encoded into the corresponding pixel in its nearest plane according to their index.

After the above procedures, a **multiple plane representation** of the input 3D model

is sorted, in which planes are in the grid form i.e. saved as 2D array, and each binary entry $\in \{0, 1\}$ represent whether it is occupied by an input point so that the multiple plane representation can be considered as a list of sub-images. Subsequently, the sub-images are recorded as holograms individually by diffraction simulation methods, e.g. Fresnel approximation or angular spectrum. In the end, the sub-holograms are superimposed together by addition operation resulting in the overall digital hologram of the input 3D scene.

Specifically, for our vectorized implementation, we normalize the depth channel such that $z \in [0, 1]$, and multiply the normalized depth values by the number of layers s.t. $\hat{z} \in [0, N]$ where N is the number of layers. Then, after rounding the adjusted depth values, the processed depth values directly become their layer indices. Similarly, to encode the pixel to its corresponding pixel in the nearest plane, the normalization-and-multiplication routine is employed again to each XY-coordinates with respect to the resolution, i.e. height and width. Next, by the advanced indexing function of Numerical Python (NumPy), all the occupied pixels are modified to 1 together without using any loops.

The number of layers and the resolution of each plane can be adjusted for the tradeoff between quality of the overall hologram and runtime of the algorithm. Also note that the double-size zero padding is applied before light propagation to avoid the circular convolution errors.

As shown in Figure 3.2, the identical squares are placed in different parallel layers. The first plot is the front-view image of four squares. The second one is the overall hologram of four layers. The other plots are the reconstructed images through focusing on different layers. Since the view direction is single and directly perpendicular to the object planes, there is no parallax, and the four squares are presented identically in the front-view image, which gives no perception of depth and focus. However, as for the encoded hologram, the detail of different squares are shown as focus distance varies; therefore, the depth cues are provided.

	Point-based method	Layer-based method
time complexity	O(number of points)	O(number of layers)
input parameter	500000 points	50 layers
sample runtime	>> 10 minutes	20.4 seconds

Table 3.1: Comparison of time complexity between point-based and layer-based methods.

As shown in Table 3.1, to encode the same 3D input point cloud with 500000 points to a 1024×1024 (padded) hologram, the runtime is decreased significantly with our proposed layer-based method. More importantly, this difference will become even more prominent

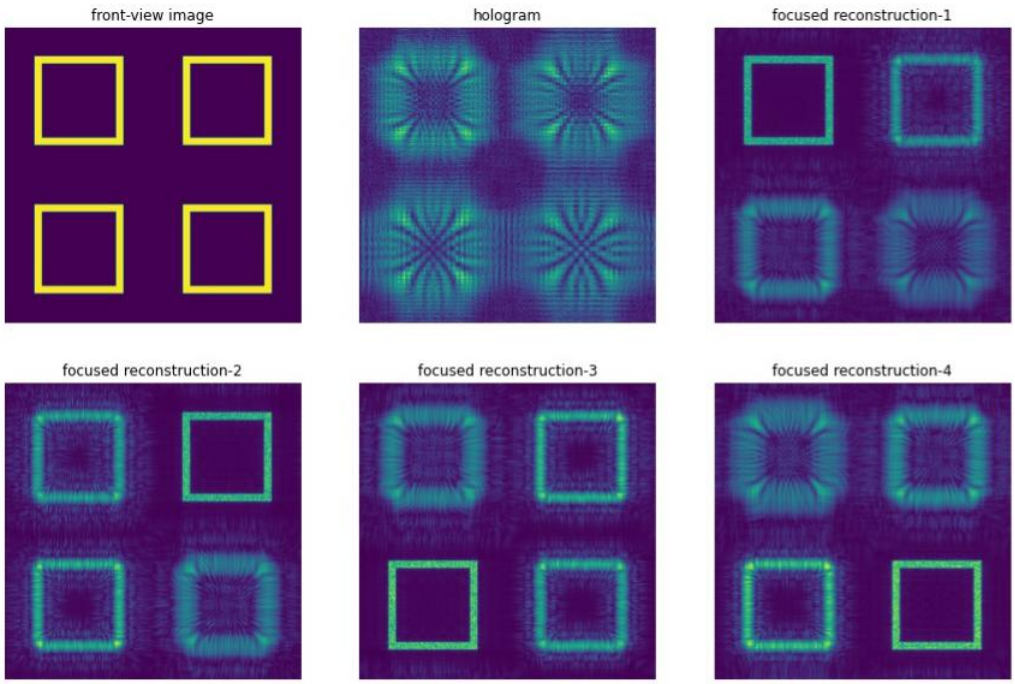

Figure 3.2: Visualization of simple sample of multi-layer CGH generated by Angular Spectrum.

as the model complexity increases because the time complexity of both methods is linearly proportional to the primitives, i.e. points or layers. Besides, note that the runtime of our proposed layer-based method excluding the diffraction simulation takes less than 1 second, which shows excellent benefits from the aforementioned vectorization process. Hence, the layer-based method has the natural advantage to be extended to high-resolution for complex 3D objects.

However, despite the efficiency of our layer-based method, there are a few shortages regarding the point-cloud-based CGH. Firstly, the number of points must be large enough to cover the whole occupied area so that there will be no tiny empty areas in its multiple plane representation. This is essential, especially when the number of layers is less because the light waves leaking through the small holes will damage the diffraction simulation of the neighboring region. Also, the texture information, e.g. small protuberances, will be rejected without a vast number of layers. The less the number of layers, the more details around the surface will be sacrificed. To fix this issue, we will introduce a solution called depth-fused 3D (DF3D) for fixing this issue later in this section.

Due to the outbreak of Covid-19 during our research, we cannot access a lab to get the optical reconstructed image of the corresponding CGHs by optical experiments. But, we

believe that the computer-generated reconstructions could still give a clear illustration of the experimental results.

In the following few subsections, to increase the precision of CGHs and make the reconstructions more realistic, we will discuss the additional features we added to our layer-based method including improved diffraction simulation method, DF3D, occlusion mask, and phase-only transformation.

3.2.2 Angular Spectrum

As for modelling the diffraction pattern of a continuous light beam from a monochromatic field, the Angular Spectrum method (ASM) is selected in our implementation. ASM is based on discrete Fourier transform (DFT) to approximate the infinite integral. Specifically, involving several steps as listed below:

1, First, ASM discretizes the continuous infinite integral by sampling over a cross-sectional grid lying within the field;

2, Then, ASM decomposes the field into a 2D angular spectrum of plane waves that each travelling in a unique direction by taking the two-dimensional fast Fourier transform (2D-FFT) of the light field;

3, Next, the complex array after the 2D-FFT is multiplied by a propagation kernel which represents the phase change of each plane wave during its way to the target plane;

4, Finally applying the two-dimensional inverse fast Fourier transform (2D-IFFT) of the complex grid to generate the target light field over the prediction plane.

Moreover, this process can be summarized into the equation:

$$H_{target}(u,v) = FFT^{-1}\{H_F(u,v)FFT(L)\} \tag{3.1}$$

$$= FFT^{-1}\{exp[i*kz*\sqrt{1-\lambda^2u^2-\lambda^2v^2}]FFT(L)\} \tag{3.2}$$

$$\approx FFT^{-1}\{exp[i*kz*\sqrt{1-\lambda^2(\frac{m}{L})^2-\lambda^2(\frac{n}{L}))^2}]FFT(L)\} \tag{3.3}$$

where Equation 3.2 and 3.3 represent the continuous and approximated discrete form respectively; u and v are the spatial frequencies, i is the complex root, k is the wave number, z is the propagation distance, L is the length of the field, $\frac{m}{L}$ and $\frac{n}{L}$ represent the grid location. Note that before the ASM, zero padding is required to avoid circular convolution errors.

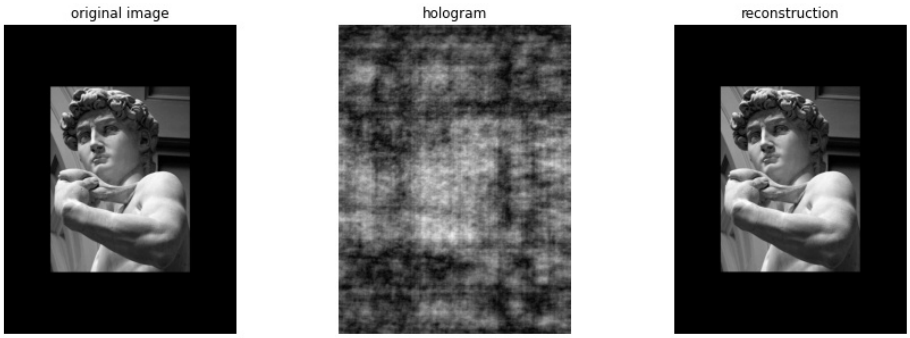

Figure 3.3: Visualization of CGH and its reconstruction. The light field propagation is simulated by a computational optics method called ASM twice.

Within the paraxial approximation, AS is equivalent to the Fresnel diffraction approximation. Although compared to Fresnel hologram, in high numerical aperture (NA) systems or close propagation distance, AS has tested to have a more stable and reliable performance [37]. Therefore, the ASM is chosen to simulate the light propagation in our method instead of standard Fresnel approximation.

To further improve the precision, we employ the band-limited ASM [32] to modify the sampling problem that existed in the original ASM. Finally, all the computations are performed via a scientific computing package for optics and visual perception called Odak [33]. The GPU acceleration is accessed through using PyTorch tensors.

3.2.3 Depth Fused 3D

By using the layer-based method, the detailed depth information of points is sacrificed for the computing speed. To recover the depth of input points and the precision of our layer-based method, the depth information is encoded to the amplitudes of light waves in the complex light field array.

$$A = A_1(z) + A_2(z) = w_1(z)A + w_2(z)A \tag{3.4}$$

$$= \frac{z_1 - z}{z_1 - z_0}A + \frac{z - z_0}{z_1 - z_0}A \tag{3.5}$$

As shown in the equations above, the amplitude of light wave from the original pixel A can be decomposed into the amplitudes of light waves from the pixels of the front and rear

layers A_1 and A_2. Moreover, the $w_1(z)$ and $w_2(z)$ are the depth-fusing weight functions that control the amplitudes in the front and rear layers, where z is the depth of the fused point. In our case, $w_1(z)$ and $w_2(z)$ are defined as the normalized distances from the original point location to the front and back layer respectively. Note that the weights are inversely proportional to the distance to layer, and $w_1(z) + w_2(z) = 1$. All the original amplitude modules in the light field are set to 1, i.e. $A = 1$.

The idea is inspired by a 3D display technology that could give the audience the near true 3D perception by using only a few planar display mediums. Since the layer structure proposed in [39, 40] shares the same principle as the slicing process of our layer-based method, the DF3D feature is transferred to our layer-based method in order to improve the accuracy of the resulted hologram. More importantly, with the assistance of DF3D, the layer-based method is able to render a hologram of a continuous 3D scene with discrete point set input. Note that in our implementation, the term luminance of the fused pixel in original papers is changed to the amplitudes of light waves in the complex field, but the luminance intensity is proportional to the square of the amplitude. However, the amplitude modules in our case are all around 1, so it is valid to assume a linear proportional relationship between luminance and amplitude.

3.2.4 Occlusion Mask

In order to simulate the diffraction pattern more realistic, the occlusion mask is also added to our layer-based method.

Generally, a good occlusion test involves either point-wise detection or extra ray casting test, any of these operations will significantly slow down the algorithm. Also, there is no commonly recognized reliable algorithm to perform occlusion processing both fast and accurate. Hence, to achieve higher efficiency, the precision of the occlusion test must be sacrificed. Instead of exhausting point-wise searching, we perform a much efficient occlusion method that progressively rejects the blocked pixels in each layer. In short, a binary mask is used to record the locations of occlusion points accumulatively and block the light from the rear points with the light field propagation.

In the beginning, a mask with all entries equal to one is built for performing subsequent light blocking. Each binary entry of the mask represents whether there is a point in front of the current pixel location, which blocks the light transmission. Then, for each layer before the light propagation, the algorithm multiplies the object grid by the mask to perform light occlusion. In the meantime, the corresponding position on the mask of the occupied

pixels on the current layer will be assigned to zero.

However, we assume that all the surfaces of the input 3D object are general and smooth, i.e. no deep valleys or high bumps, also the light source are far enough to ignore the blocking angle. Based on these assumptions, only employing a binary mask iteratively is sufficient to lead to an acceptable result. Although, our occlusion mask is very fast and easy to implement in practice.

3.2.5 Phase-only Hologram

Since the available spatial light modulators (SLMs) can only support phase-only input to provide an end-to-end computer-generated holography routine, another critical problem is transferring the complex hologram to the phase-only hologram (POH).

To begin with, the most straightforward method should be removing all the amplitudes directly, the quality of the corresponding reconstruction will be damaged significantly. Hence this approach can only be applied to some simple shapes. To recover the error caused by amplitude removal, the Floyd-Steinberg error diffusion method proposed in [41] is employed. The basic idea is passing the errors from forcing the values of the complex hologram to have unit amplitude to the neighboring pixels iteratively. Firstly, we define:

$$|H_p(u,v)| = 1 \tag{3.6}$$

$$arg(H_p(u,v)) = arg(H(u,v)) \tag{3.7}$$

$$E(u,v) = H(u,v) - H_p(u,v) \tag{3.8}$$

where $|\cdot|$ and $arg(\cdot)$ denote the modulus and argument of the complex number respectively; H is the initial complex hologram; H_p is the amplitude removal hologram; E is the differences between H and H_p i.e. the error term.

Then, the pixels in the hologram is processed following the top-to-bottom and left-to-right order, according to the equations below.

$$H(u, v+1) \leftarrow H(u, v+1) + w_1 E(u, v) \tag{3.9}$$
$$H(u+1, v-1) \leftarrow H(u+1, v-1) + w_2 E(u, v) \tag{3.10}$$
$$H(u+1, v) \leftarrow H(u+1, v) + w_3 E(u, v) \tag{3.11}$$
$$H(u+1, v+1) \leftarrow H(u+1, v+1) + w_4 E(u, v) \tag{3.12}$$

Despite the clarity and efficiency of the method above, the effects of noises will be accumulated by passing the errors to neighboring pixels in an iterative manner. To solve the correlation between noisy pixels, we apply the enhanced version of error diffusion proposed in [42] i.e. the bidirectional error diffusion process.

More than the operations in [41], [42] processes the even rows based on the previous equation sets, but for odd rows, each pixel is processed based on another set of equations.

$$H(u, v-1) \leftarrow H(u, v-1) + w_1 E(u, v) \tag{3.13}$$
$$H(u+1, v+1) \leftarrow H(u+1, v+1) + w_2 E(u, v) \tag{3.14}$$
$$H(u+1, v) \leftarrow H(u+1, v) + w_3 E(u, v) \tag{3.15}$$
$$H(u+1, v-1) \leftarrow H(u+1, v-1) + w_4 E(u, v) \tag{3.16}$$

In other words, each row of the complex hologram is processed from opposite directions alternatively. Compared to the unidirectional error diffusion, the bidirectional version decomposes the correlation between noises to achieve better quality.

However, similar to all the iterative methods, the time complexity $O(mn)$ (where m and n represent the resolution) of the error diffusion method is high and cannot be vectorized in the programming implementation. Although, the error diffusion is easy to implement and clear to understand.

As shown in the graph, the reconstruction quality of POH outperforms the amplitude removal method but is still not as good as the original complex hologram. Due to the rapid development of deep learning, neural networks are now employed widely in the field of phase retrieval, and phase recovery [43, 44]. The precision and speed of such method are promising. However, the additional training time and large data requirement make deep learning hard to apply in practice. In this case, without a corresponding public dataset, since the phase-only transformation is not our primary focus of this section, we accept

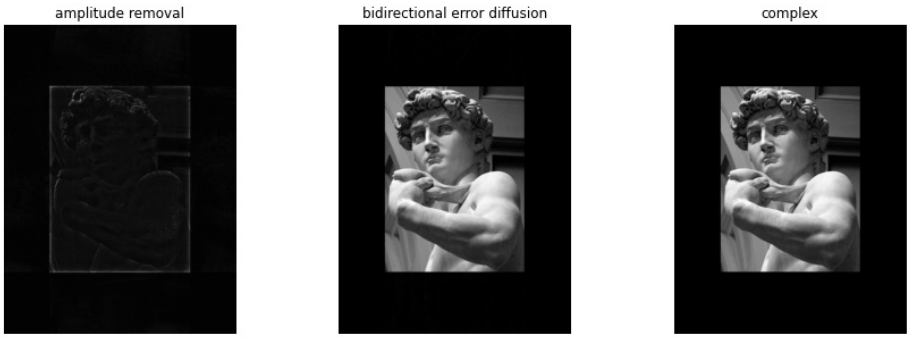

Figure 3.4: Comparison of different complex to phase-only methods.

error diffusion as a satisfying alternative.

3.3 3D Hologram Reconstruction

In this section, we propose a novel learning-based point cloud reconstruction method from digital holograms.

Firstly, for image-based hologram reconstruction with known focal distance, one single light propagation is enough to reproduce the original image. Unlike the 2D planar image reconstruction, the 3D reconstruction from a digital hologram requires several runs from different focal distances to acquire the details from different parts of the original object. Therefore, the 3D reconstruction process must be adaptive to locate the in-focus areas from different focal distances. For simplicity, the extra features, e.g. RGB color channels, are not considered in our case and focusing on the 3D geometry reconstruction.

To extract a point cloud from a digital hologram, focusing at multiple depths is the crucial stage. Similar to this idea, as proposed in [45, 46, 47, 48], auto-focusing is the main topic for distance recovery of the hologram. More importantly, due to the predicting nature of auto-focusing, it is easy to relate and fuse auto-focusing with deep learning. For instance, in [45], the holograms are recorded from five discrete distances, then predicting the distance between the object image plane and recording plane is considered as a classification task for each hologram. So, a convolutional neural network (CNN) is trained to predict the distance label of input digital hologram through forward propagation. To improve this algorithm, [46, 47] is proposed so that the depth prediction is considered as a regression problem, and a CNN with two fully connected layers is able to predict continuous focal distance.

However, the current proposed auto-focusing technologies are mainly discussed under the 2D reconstruction situation. Hence, based on the ideas mentioned above of learning-based auto-focusing and the layer-based computer-generated holography, we propose a novel reversed layer-based routine with deep learning to extract 3D geometry from the stereoscopic holograms.

3.3.1 Reversed Layer-based Reconstruction Method

This subsection is for the detailed explanation of our reversed layer-based reconstruction method.

At the beginning, similar to the layer-based encoding method, the input digital hologram is reconstructed through multiple focal distances by the light propagation approximation, resulting in a list of reconstructed sub-images. And for simplicity, the depth values $\mathbf{z}$ of the sub-images are assumed to be known as prior information.

Since the in-focus pixels are equivalent to the points in the initial point cloud with known depth z, we need to locate the missing location, i.e. XY-value of each in-focus pixel in the sub-images, to reproduce the original 3D geometry. We find that this task can be done through image segmentation, i.e. each pixel of the sub-images need to be classified according to whether it is in-focus. Hence, a deep segmentation neural network is trained to perform the aforementioned binary segmentation via forward propagation for each sub-images. In the end, a post-processing step is added to fill the necessary gaps between layers by linear interpolation so that a continuous surface is constructed.

The brief structure and relationship of our two CGH algorithms are shown in the Figure 3.5, where **(a)** represents the multiple plane representation of input point cloud; **(b)** represents the diffraction simulation results corresponding to each plane; **(c)** represents the list of reconstructed sub-images; **(d)** represents the multiple plane representation of reconstructed points.

Note that due to the occlusion mask, the input 3D model is reduced to only the front surface, and the accuracy of light field propagation approximation may decrease as the reconstruction distance increases. These two factors would have a large possibility of affecting the reconstruction precision, thus needing to be noticed in the following experiments. Also, the results of reconstruction will depend significantly on the quality of the input hologram. For holograms with detailed depth cues, the reconstructed point cloud should have high precision. However, without a public hologram dataset including carefully tagged depth indicators, due to experimental errors in data preparation, our training neural network

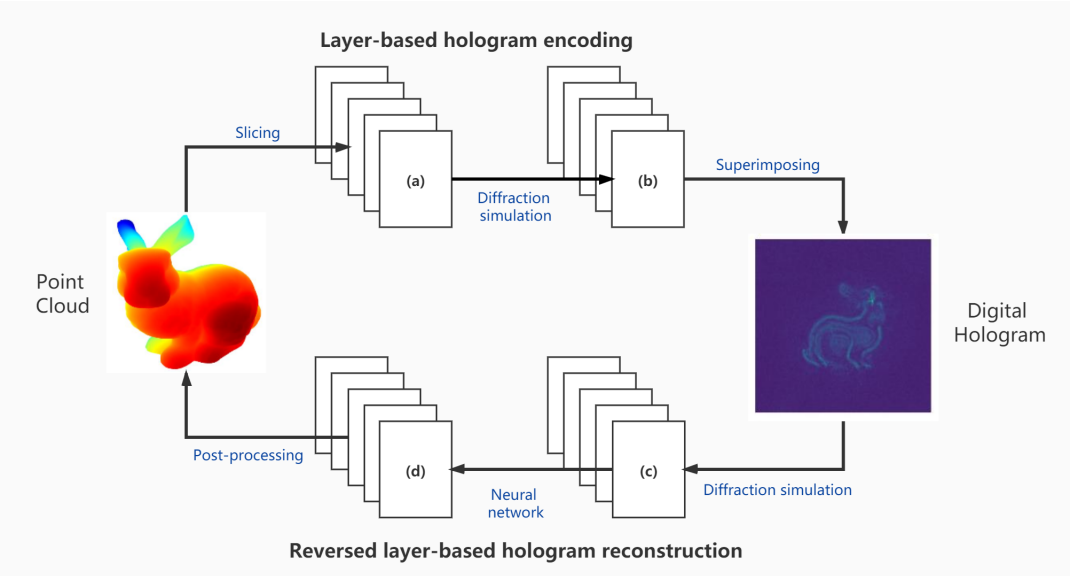

Figure 3.5: Visualization of the symmetric layer-based 3D hologram encoding and reconstruction methods.

may also be biased.

More importantly, there are several reasons that push us to make the above innovations. First, in the past deep learning fused auto-focusing method, either conventional CNN or vanilla U-net network [49] is adapted to hologram depth estimation tasks [45, 46, 48]. Therefore, we would like to improve the result by upgrading the network architecture, i.e. switch the network to a more powerful variant of vanilla U-net. Second, compared to the numerical metric-based methods, a trained neural network is used to enhance efficiency. Finally, in the post-processing stage, linear interpolation is used around the connection between layers in order to smooth the output 3D surface. More importantly, our proposed point cloud reconstruction method can be extended to three-dimensional auto-focusing that focuses on 3D scenes adaptively. And since the 3D auto-focusing task is equivalent to extracting a depth map from the hologram, we believe that solving 3D auto-focusing would be convenient for many downstream applications.

In summary, our proposed 3D reconstruction method could bridge the gap between the digital hologram and 3D geometry in terms of the point cloud. Since point clouds work as the most fundamental 3D data representation, linkage with point cloud enables digital hologram to more extensive applications. In the following subsection, we will talk about the deep learning network architecture employed in our implementation.

3.3.2 Network Architecture

U-net

As the very first successful neural network architecture which has shown remarkable performance on medical image segmentation tasks, U-net is one of the best candidate models to be adapted to the segmentation process in our reversed layer-based 3D reconstruction.

Before [49], solving vision tasks by the convolutional neural network (CNN) once became a top trend, but most applications of CNN were restricted in the field of image classification, i.e. producing a single class label for each input image. However, to extend the usage of CNN, [49] proposed using a U-shape CNN to perform image segmentation task, i.e. generating the per-pixel class label based on the input image. The U-shape means that the network is symmetric and consists of a contracting path and an expansive path. The U-net behaves like conventional CNN in the beginning, the contracting path uses convolution filters with non-linear activation function and max-pooling to down-sample the feature values and doubles the feature channels. Then, instead of passing through the final features to a fully connected layer to gain the classification probability vector, the U-net applies an expansive path subsequently.

$$\{C\}_{ij} = c_{ij} = \mathbf{w} \cdot \mathbf{X}_{i:i+m-1,j:j+m-1} + b \tag{3.17}$$

$$C = ReLU(\mathbf{C}) = max(\mathbf{C}, 0) \tag{3.18}$$

$$\hat{C} = \text{Max-pooling}(C) \tag{3.19}$$

The equations above represent the operations inside a convolutional block in the contracting path, where $\mathbf{w}$ denotes the convolutional kernel with size m; X is the input tensor and c_{ij} is the extracted feature. And ReLU$(\cdot)$ is the non-linear activation function, Rectified Linear Unit (ReLU); max-pooling$(\cdot)$ represents the max-pooling operation, where $\hat{C}$ is the pooled feature map.

After the contracting path, U-net sends the features to an expansive path which uses up-convolutions to half the number of feature channels and up-sample the feature values. In addition, to recover the boundary information lost from the previous convolution operations, the cropped feature maps from the contracting path are concatenated to the corresponding stages in the expansive path. For the final layer, compared to the regular CNN, the fully connected layer is replaced by a 1×1 convolution layer that maps the

ultimate features to the per-pixel class labels.

In conclusion, U-net can create the corresponding segmentation map for the input image by adding an expansive path. Besides, based on the network structure shown in the Figure 3.8, if we remove all the attention gates, as well as replacing every recurrent block and shortcut connection by double elementary (up-)convolutional layers and copy-and-corp operation respectively, the resulted plot is precisely for the vanilla U-net architecture.

R2 Attention U-net

Due to the great success of U-net in medical image segmentation, there are several variants [50, 51] of U-net were introduced subsequently. In our case, we would like to highlight two of them in detail and merge them into our method. The first one is Recurrent Residual U-net (R2U-net) [50].

As its name suggests, R2U-net utilizes two appropriate features from relevant works into the original U-net. Firstly, since the recurrent neural network (RNN) achieved outstanding performance in handling the ordered sequence data, e.g. sentences in the natural language processing (NLP) [52], the recurrent convolution is proposed to computer vision subsequently [53]. Compared to the conventional convolution layer, recurrent convolution layer receives and calculates the input from several different time steps so that enhancing the feature extraction, especially for the features from a very low level. Therefore, the loss of details from the original image will be reduced, resulting in a better feature representation which benefits the final result. Also, R2U-net uses the shortcut connection from the residual neural network (ResNet) [53]. The main advantage of shortcut connection is that it ensures a faster and better convergence without losing the depth of the network. More importantly, since both the recurrent and residual operations do not add new parameters to the network, R2U-net improves the performance without expanding the size of the network. As shown in the Figure 3.6, the combination of recurrent convolution and shortcut connection forms the recurrent residual convolutional units (RRCU) proposed in [50].

The next improved U-net is the attention U-net [51]. [51] suggested that adding the attention gate to control the focus of the network during the expansive path of U-net. The introduction of attention mechanism is very intuitively reasonable, because for many cases in image segmentation, especially for medical images which consist of many tiny cells or tissues, the target areas are tiny; a compact method is needed to restrict the focus of the network to the most possible region over the whole image, and one of the best solutions is the attention gate. The brief visualization of each attention gate is shown in the Figure

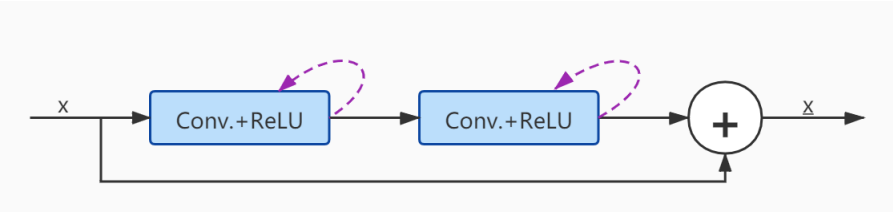

Figure 3.6: Inside the RRCNN unit in R2U-net. Same as the recurrent convolution block used in R2 attention U-net.

3.7, where x represents the input features, g denotes the gating signals; and W_g, W_x, W_a correspond to the coefficients of linear transformations with bias term. After combining the three linear transformations with activation functions, the resulting shrank array is passed through a grid trilinear interpolation resampler to form the complete element-wise attention coefficients. Finally, the attention gates multiply the attention coefficients by the feature map to emphasize the important features and ignore the irrelevant features. In conclusion, to cover a larger perceptive field, the attention gates aim to produce the attention coefficients $\alpha_i \in [0, 1]$ as the saliency map to force the network to focus on only the pixels relevant to the segmentation task. The result of attention U-net is impressive, which makes the idea of attention more convincing.

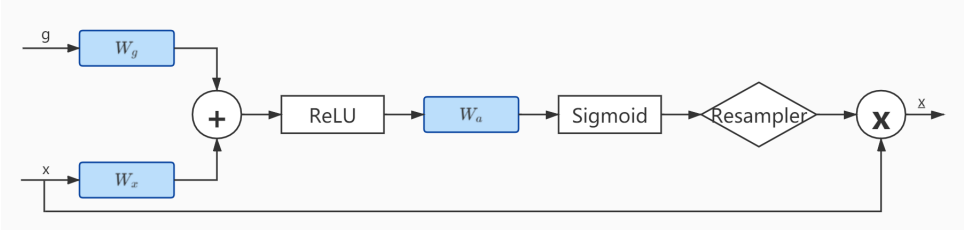

Figure 3.7: Inside the attention gate in R2 attention U-net.

Since the above features, including the RRCNN unit and attention gate, are independent of each other, to enhance the performance of the image segmentation in our case, we will fuse them and the U-net to form the novel Recurrent Residual Attention U-net (R2 Attention U-net) as shown in the Figure 3.8.

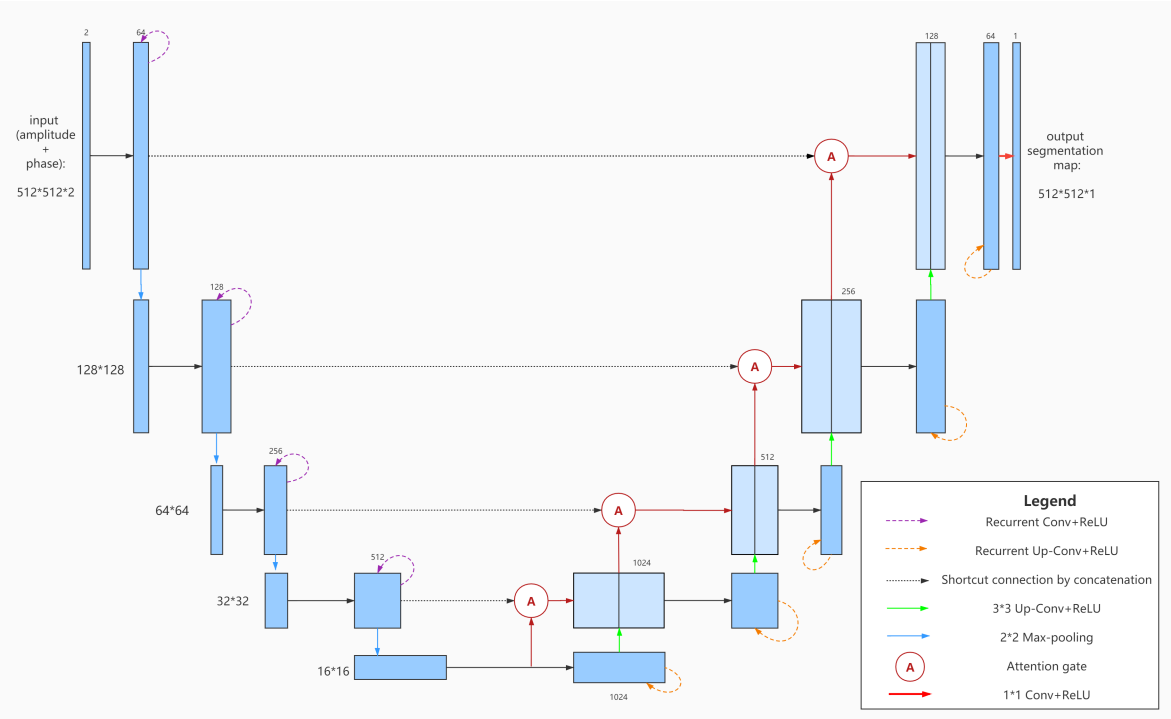

Figure 3.8: Visualization of the Recurrent Residual Attention Convolutional Neural Network based on U-net (R2 attention U-net).

3.3.3 Experiments

We perform the experiments about the layer-based method for constructing CGHs in section 3.2.1 and the 3D reconstruction algorithm of extracting point cloud from the given digital hologram in section 3.3. The experimental setup and results are shown in the following sections, including the subsequent detailed analysis and reasoning.

Basic Settings

The basic settings of our experiments are shown in Table 3.2. First, for the number of points in the input point cloud, 5000000 points is enough to capture the complete 3D geometry information for any common surfaces. However, this number may be modified to ensure that the point cloud is dense enough to perform compact occlusion as discussed in subsection **3.2.4**. The basic settings, i.e. recording distance and pixel pitch, are selected initially as prior information.

Then, for the number of layers and the distance between layers, note that their numbers in the Table 3.2 are an example of their suitable combination, and these numbers are not

fixed in practice. In other words, the number of layers is determined first, and the distance between layers is considered as a hyper-parameter that to suit the former choice to improve the quality of resulted CGH. The basic idea is to control the total length of the model, i.e. number of layers $\times$ distance between layers in the near field and match the pre-defined pixel pitch and recording distance. Additionally, we choose to use the light with wavelength 650 nm, since the higher the light's wavelength λ, the longer the diffraction pattern is so that the more details could be extracted.

Parameter	Value
wavelength	650 nm $= 6.5 * 10^{-7}$ m
wave number	$\approx 9.7 * 10^{6}$
distance	0.15 m
pixel pitch	8 μm $= 8 * 10^{-6}$ m
resolution	1440×1440 (padded)
number of points	5,000,000 $= 5 * 10^{6}$
number of layers	10
distance between layers	4 mm $= 4 * 10^{-3}$ m

Table 3.2: Experimental settings.

Data Preparation

Since there is no public digital hologram dataset with multiple depth indicators available, we build a personal dataset for training and testing the neural network.

In this case, we employ the layer-based hologram encoding method discussed in the previous section **3.2.1** to construct the dataset. The multiple plane representation of each input point cloud (**(a)** in the Figure 3.5) and their corresponding sub-reconstructions (**(c)** in the Figure 3.5) are saved in pairs, becoming the input data and ground truth. In particular, we set the total layer number equals 200. In this case, the number of occupied pixels in each layer is very small, so we reconstruct the hologram every 20 layers, and adding the in-focus regions of these layers together to form the corresponding segmentation map.

Then, we select a total of 60 3D-CAD models from ShapeNetCore V2 [56], 42 for training our proposed neural network, and 9 additional objects each for validation and testing. The candidate 3D models are chosen carefully so that the complexity of each is intermediate, i.e. preventing the model from overfitting and maintaining the difficulty. Originally, each CAD model is saved as 3D mesh and in **.obj** format, so the first step is translating 3D meshes to point clouds by sampling.

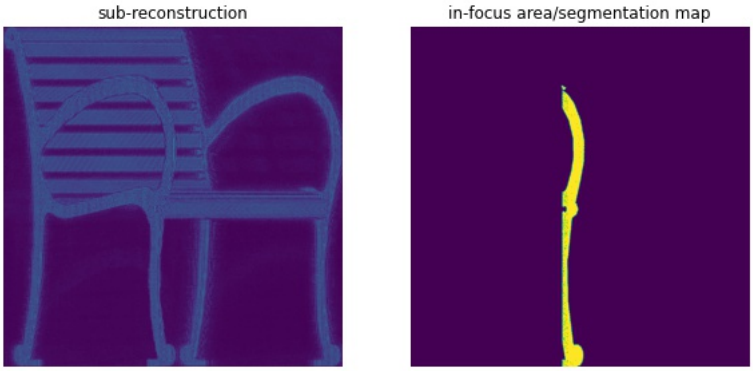

Figure 3.9: The input x_i and ground truth y_i of our dataset.

As for data augmentation, since the sampled 3D models are not symmetric, all the samples are recorded in 3 perpendicular directions plus a random degrees $\theta \in [0, \pi/4]$ with respect to XYZ-axes respectively to enlarge the available dataset. Overall, we get 1260 pairs of training data, and 270 pairs each to validate and test the trained network. However, due to the occlusion mask, there will be tiny in-focus area in the rear layers, so we prune the data instance that the ratio of in-focus region is less than 1% of the total area to avoid overfitting.

To further improve the data effectiveness, the amplitude and phase information is extracted separately from each sub-image. The first reason is that real values are easier to be saved and read, and preventing passing through complex input to the neural network could avoid unnecessary complexity and errors. And the second is that applying real convolution operation with respect to amplitude and phase is more understandable and explainable.

In the end, due to the instability of the UCL cluster and Google Colaboratory (from overseas access) and local computing power limitation, unfortunately, the sizes of input and output are reduced to 256×256, and we fail to try a larger training set.

Training Setup

The loss equation we used in the training process is the Binary Cross Entropy Loss (BCE Loss).

$$Loss_{bce}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{m}\sum_{i=1}^{m}[y_i * log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i)] \tag{3.20}$$

$$= \frac{1}{m}\sum_{i=1}^{m}[y_i * log(f_\theta(x_i)) + (1 - y_i)log(1 - f_\theta(x_i))] \tag{3.21}$$

where x_i and y_i are the input and target respectively; m denotes the number of samples in a mini-batch; f_θ denotes the neural network, and θ is its parameters, $f_\theta(x_i) \in [0, 1] \forall i$. In our implementation, the batch size m is set to be 2.

Next, the Adam optimizer [57] is applied for optimizing the network. The Adam is combining the adaptive learning rate ideas with momentum, thus the convergence speed of Adam and quality are both better than almost all other optimizers. The mechanism of Adam is shown by the equations below:

$$m_{t,i} = (1 - \beta_1)m_{t-1,i} + \beta_1 g_{t,i} \tag{3.22}$$

$$s_{t,i} = (1 - \beta_2)s_{t-1,i} + \beta_2 g_{t,i}^2 \tag{3.23}$$

$$\theta_i^{t+1} = \theta_i^t - \frac{\alpha m_{t,i}}{\sqrt{s_{t,i} + \epsilon}} \tag{3.24}$$

where t is the current iteration index; i is the index of optimizing parameter (θ); m and s can be considered as two momentum terms, and β_1 and β_2 are their hyper-parameters; ϵ is an arbitrary small number; α denotes the learning rate.

In our implementation, we follow the settings given in the original Adam paper [57], i.e. $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\alpha = 0.001$, and $\epsilon = 10^{-8}$. Furthermore, the number of epochs is 150 to fully study the training data and reduce trivial training runtime. And, after the 75-th epoch, the learning rate α starts decreasing to ensure the convergence of the network, and the learning rate will reduce to 0 in the last epoch.

Furthermore, since the parameter initialization is extremely important for training neural networks, to ensure a faster and stabler convergence, we select the initialization strategy to the Kaiming initialization [58]. Because Kaiming initialization is explicitly developed for the CNNs with ReLU as its activation function.

All the deep learning-related procedures are done with the help of Python library: Pytorch [59]. Besides, the number of workers is set to 4 in practice enabling hardware to total usage.

3.3.4 Results and Discussion

Firstly, for the layer-based hologram construction method, as shown in the Figure 3.10 and 3.11, the results are promising, i.e. the details of the input point cloud are successfully encoded into the digital hologram, and focusing towards different focal distances ideally gives the depth cues.

To be specific, as shown in Figure 3.10, the detailed image of different chair legs and arms are presented as focal distance goes further, and the rest of the chair is blurred slightly to reveal the right in-focus region. Also in the Figure 3.11, the quality of the encoded hologram has strengthened with the increasing number of encoding layers. When the number of layers $N = 100$, the input point cloud is encoded into the hologram with a relatively coarse level of details, however the result is acceptable except few taints; when $N = 250$, the level of details becomes very high, particularly the texture around the hind leg of the bunny is recorded. The results demonstrate that our proposed layer-based method can encode the three-dimensional geometry of the input point cloud detailedly into a digital hologram where the depth cues appear correctly via focusing. However, to show the details of the object that are preserved in the encoded hologram, we lower the distance between layers so that the overall structure of the reconstruction can be more clear. To increase the effect of focus cues, we can simply raise the distance between layers during the encoding process, and the in-focus area will be more significant.

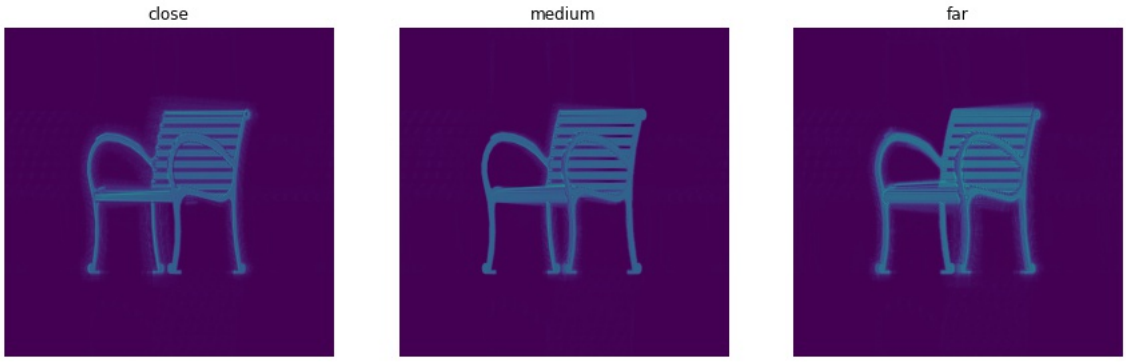

Figure 3.10: Reconstructions from different focus.

In addition, to indicate the improvement regarding the introduced features, we test the performance of our method without depth-fused 3D (DF3D) or occlusion mask as shown in Figure 3.12. Compared to the first plot in Figure 3.11, without the DF3D, the reconstruction can only give an outline of the bunny, the depth cues are wholly rejected, this difference is more noticeable when the layer number is low; and without the occlusion

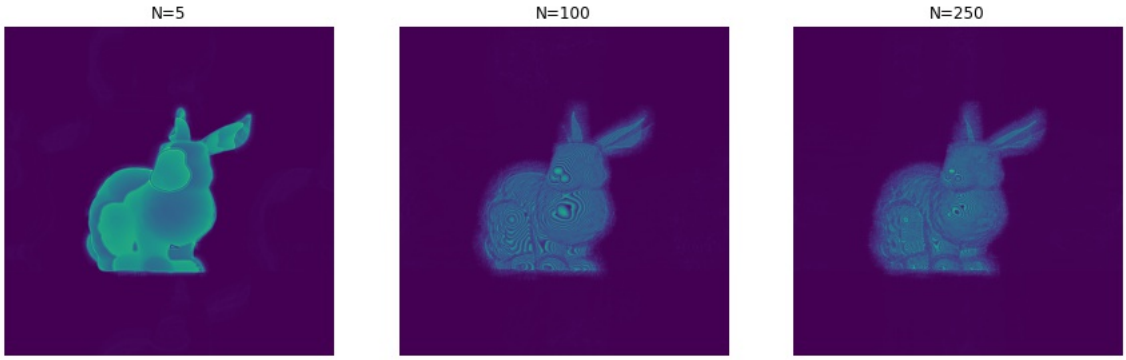

Figure 3.11: Reconstructions from different number of layers.

mask, the light fields of the rear layer and the close layer are cancelled out, resulting in an ambiguous geometry.

Figure 3.12: Reconstructions from layer-based method without DF3D or occlusion mask.

Furthermore, we have shown the efficiency of our layer-based method compared to the point-based method in Table 3.1. To further illustrate the improvement, we test the runtime given the number of layers equals 250 with resolution 1024×1024 (padded), the algorithm only takes an average of 106 seconds on a 2.5GHz Intel-i5 CPU without GPU parallel computing. Since our method is only based on the primary operations of NumPy, the GPU acceleration can be easily accessed by transferring the NumPy operations to the corresponding PyTorch operations or any CUDA/JIT-involved numerical libraries.

Next, for the learning-based point cloud reconstruction method, as shown in Figure 3.13, the training loss keeps decreasing until the $110 - th$ epoch and stabilizes around 10.

However, even the losses converge normally, the performance of the trained neural network is not as good as we expected. We find that the neural network can only identify and segment the extensive and continuous in-focus area, and the prediction would become

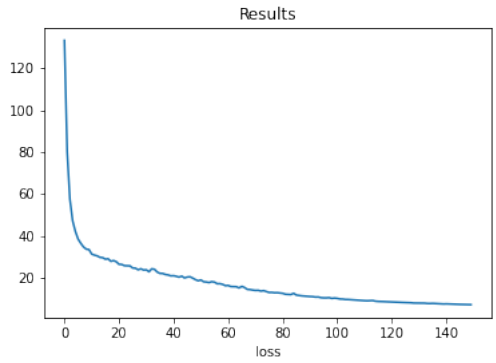

Figure 3.13: Training epoch loss versus epoch number.

ambiguous if the in-focus area is separated into tiny fragments over the whole image. We conclude that the failure is from these major aspects:

1. Unlike the medical image segmentation, where the shapes of the target objects (e.g. cell, blood vessel) are continuous and similar, the shape of the in-focus area in a reconstructed hologram is primarily random and discrete. More importantly, since our dataset is built of single 3D objects and the focus cues are determined by focal distance, the target area is often scattered over the segmentation map, which significantly increasing the difficulty of the task.

2. The transition between the in-focus area and the out-of-focus area is smooth in our dataset. Rather than a clear boundary between in-focus and out-of-focus regions, the shift between them is gentle in our dataset, so it is hard for the network to learn a precise decision boundary to separate the in-focus areas. This is because we consider the size of each object is small, the focus cues are not significantly distinguishable for every single 3D object (as shown in Figure 3.9, the out-of-focus region is blurred mildly). However, if we consider the size of the object is large and increase the distance between layers, the focus cues will be more apparent, and the difficulty of the segmentation task will decrease accordingly.

3. The quality of the custom hologram dataset is questionable. Firstly, the hologram resolution is discounted because of the hardware limitation, 256×256 resolution is not enough to capture all the details of the input point cloud so that the accuracy of the encoded hologram is reduced. Secondly, the precision of our layer-based encoding method still has space to improve compared to the brute-force simulation, especially given the relatively low resolution. Thirdly, there are some inherent defects of the holography, e.g. the edge fringe effect [60], which also cause harmful effects to the resulted hologram, resulting in

the outliers and reducing the generalization ability of the network.

To address the issues mentioned above, we propose the following adjustments:

1. lowering the difficulty of the task, i.e. enhance the degree of the depth cues in the reconstruction by increasing the distance between layers while building the custom dataset. Then, it will be easier to distinguish the points from different layers, and the transition between in-focus and out-of-focus area will be more apparent for the neural network to classify.

2. adding one more layer or pre-processing stage that using numerical method to derive the sharpness of the input image. Instead of receiving amplitude and phase information, additional sharpness information will help the segmentation task become more concrete.

In conclusion, a better hologram dataset is needed for a better convergence of the proposed neural network. Also, the result of our proposed architecture is likely to become promising if we compromise the quality of the custom dataset, i.e. strengthen the perception of depth by increasing the distance between layers but sacrifice the perception of the overall structure of the 3D objects.

Chapter 4

Conclusions and Future Works

In this paper, as for our main contribution: the symmetric layer-based hologram encoding and decoding pipeline, there is still space to progress. In particular, the results of the encoding process, i.e. the quality of resulted digital hologram, is encouraging. The details of the original point cloud is well preserved in its hologram representation, also the speed of our method is also much faster than the general point-based method. To further enhance the performance of the encoding method, few potential developments could be implemented, and we will list them in the following section. However, for the reconstruction process, the results are not very satisfying, and we will also discuss the possible solutions in the next subsection.

In addition, by comparing the state-of-the-art compression methods, we notice that both the compression rate and speed of these algorithms are promising. The corresponding mesh compression method has already been used in 3D modelling software and Web applications. However, since the current state-of-the-art point cloud compression methods are mainly learning-based, their generalization ability is questionable, therefore, cannot be adopted for public use. Furthermore, based on the survey of translation algorithms between different three-dimensional data representations, this has shown the different characteristics of each representation, In short, the point cloud is straightforward to represent three-dimensional geometry but sparse and unordered; 3D mesh has additional topological information about the surface compared to point cloud; signed distance function is lightweighted and handy for rendering but hard to derive. These characteristics make each representation suitable for specific applications. Moreover, based on the former research, we gain inspiration about the common techniques of processing the data in three-dimensional space, such as voxelization or Marching Cubes algorithm use the discretization to structure

the 3D data and enhance the efficiency; this idea is also applied in our layer-based methods.

Future works

In this subsection, we discuss the possible improvements regarding our main contributions.

Initially, although the depth-fused 3D technique is helpful to retain the depth detail of input in its multiple-plane representation, especially for a low number of layers, to enable its total usage, a numerical method for determining the relationship between amplitude modules of the DF3D fields and the layer distance is needed. With a stable equation of these two variables, the light change between DF3D layers would be more smooth and coherent so that the overall hologram will be more natural.

The subsequent adjustment is for the occlusion mask. Even though the current occlusion mask method is enough to operate light-blocking, we question its performance under more extreme situations. For instance, if the two objects are separated far away with respect to the depth $z-$axis, the light waves would diffract from the near object so that the diffraction cone covers more area of the rear object than the size of the near object. So the current occlusion mask approach will be biased in this case. However, the present method has a massive advantage in its efficiency compared to the other more precise and realistic occlusion test e.g. [38], so our next target is to find an elegant way to perform the light occlusion both physically and speedily.

Another feasible development is the shape completion procedure at the end of reversed layer-based 3D reconstruction, i.e. stitching the points from separated layers. Based on [61], one possible alternative is using one additional neural network to up-sample the reconstructed point cloud and refine the quality of the reconstructed point cloud. This idea is suitable in our case because the structures of the multiple plane representation of the reconstructed point cloud are similar, it would be easier for the network to generalize given such a specified objective.

However, our major future work should be solving the remaining problems for the point cloud reconstruction algorithm. The current biggest obstacle is the dataset, and there are two adjusting options. First, we could enlarge the length of the 3D objects from ShapeNetCore, so the depth cues in the reconstructions will be more significant. Also, since the depth cues of a hologram are given by focusing, the in-focus area will be more distinguishable by the proposed neural network in this case. Second, we could choose a more accurate method to build our dataset, e.g. deep learning-based methods [38, 44, 48],

and increasing the resolution of the input image and output segmentation map. In addition, another improvement is employing the numerical method that can derive the sharpness of each sub-region of the image as pre-processing.

In the end, we would like to argue the future usage of deep learning in the field of computational optics. It has been a big trend of employing deep learning to solve optics problems such as phase retrieval and hologram generation [38, 43, 44], this is mainly because the neural networks are able to finish the tasks accurately with a high even real-time speed. But, due to the black-box nature of deep learning, rather than applying one end-to-end neural network, we would like to emphasize the fusion of optics knowledge and deep learning e.g. the HoloNet [62]. To be specific, using the prior knowledge of optics to improve the application of deep learning, so the algorithm would be more interpretable and explainable, and the usage of deep learning could become more appropriate compared to a single end-to-end neural network. Most importantly, the large-scale hologram dataset with labelled depth indicators is urgent for the future research of holograms with deep learning.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton., ImageNet classification with deep convolutional neural networks, *In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*.Curran Associates Inc., Red Hook, NY, USA, 1097–1105. 2012.

[2] Charles R. Qi et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, *arXiv:1612.00593*, cs.CV, 2017.

[3] E. Yurtsever, J. Lambert, A. Carballo and K. Takeda, A Survey of Autonomous Driving: Common Practices and Emerging Technologies, *in IEEE Access, vol. 8, pp. 58443-58469*, 2020, doi: 10.1109/ACCESS.2020.2983149.

[4] Charles R. Qi et al. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, *arXiv:1706.02413*, cs.CV, 2017.

[5] Osher, Stanley, and Ronald Fedkiw, "Signed distance functions." Level set methods and dynamic implicit surfaces, *Springer*, New York, NY, 2003. 17-22.

[6] Siheng Chen et al., 3D Point Cloud Processing and Learning for Autonomous Driving, *arXiv:2003.00601*, cs.CV, 2020.

[7] Koulieris, George Alex, et al., Near-eye display and tracking technologies for virtual and augmented reality, *Computer Graphics Forum*. Vol. 38. No. 2. 2019.

[8] Takikawa, Towaki, et al., Neural geometric level of detail: Real-time rendering with implicit 3D shapes, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[9] Orlosky, Jason, et al., Telelife: The Future of Remote Living, *arXiv preprint arXiv:2107.02965*, (2021).

[10] Cao, Chao and Preda, Marius and Zaharia, Titus, 3D Point Cloud Compression: A Survey, *The 24th International Conference on 3D Web Technology*, 2019, pages 1-9, doi: 10.1145/3329714.3338130.

[11] S. Schwarz et al., Emerging MPEG Standards for Point Cloud Compression, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* March 2019, vol. 9, no. 1, pp. 133-148, doi: 10.1109/JETCAS.2018.2885981.

[12] Dat Thanh Nguyen and Maurice Quach and Giuseppe Valenzise and Pierre Duhamel, Learning-based lossless compression of 3D point cloud geometry, *arXiv:2011.14700*, eess.IV, 2021.

[13] Huang T , Liu Y , 3D Point Cloud Geometry Compression on Deep Learning, *the 27th ACM International Conference*, 2019.

[14] Wang, Jianqiang and Ding, Dandan and Li, Zhu and Ma, Zhan, Multiscale Point Cloud Geometry Compression, *2021 Data Compression Conference (DCC)*, 2021, pp. 73-82, doi: 10.1109/DCC50243.2021.00015.

[15] Choy, Christopher and Gwak, JunYoung and Savarese, Silvio, 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pages: 3075-3084

[16] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, *arXiv: 1602.07261*, 2016

[17] Khaled Mammou and Rémi Arnaud, Open 3D Graphics Compression (Open3DGC), *GitHub repository*, `https://github.com/amd/rest3d/tree/master/server/o3dgc`.

[18] Khaled Mamou, Titus Zaharia, Françoise Prêteux, TFAN: A low complexity 3D mesh compression algorithm, *Computer Animation & Virtual Worlds*, 2010, 20(2-3):343-354.

[19] Khronos Groups, glTF – Runtime 3D Asset Delivery, *GitHub repository*, `https://github.com/KhronosGroup/glTF`

[20] Google, DRACO, *GitHub repository*, `https://github.com/google/draco`

[21] J. Rossignac, Edgebreaker: connectivity compression for triangle meshes, *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47-61, Jan.-March 1999, doi: 10.1109/2945.764870.

[22] Rossignac J , Safonova A , Szymczak A, 3D Compression Made Simple: Edgebreaker on a Corner-Table, *IEEE*, 2001, doi: 10.1109/2001.923399.

[23] Xu Yi, Lu Yao, and Wen Ziyu, Owlii dynamic human mesh sequence dataset, *ISO/IEC JTC1/SC29/WG11 (MPEG/JPEG) m41658*, October 2017.

[24] Bernardini, Fausto et al., The Ball-Pivoting Algorithm for Surface Reconstruction, *Visualization and Computer Graphics, IEEE Transactions on*, November 1999, Pages 349-359, doi: 10.1109/2945.817351.

[25] Michael Kazhdan1, Matthew Bolitho1 and Hugues Hoppe, Poisson surface reconstruction, *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, June 2006, Pages 61–70.

[26] Lorensen, William E. and Cline, Harvey E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm, *Association for Computing Machinery*, July 1987, doi: 10.1145/37402.37422.

[27] Timothy S. Newman, Hong Yi, A survey of the marching cubes algorithm, *Computers & Graphics*, 2006, 30(5):854-879.

[28] Jeong Joon Park, et al., DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, *arXiv: 1901.05103*, 2019.

[29] Towaki Takikawa, et al., Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes, *arXiv: 2101.10994*, 2021.

[30] G. K. Wallace, The JPEG still picture compression standard, *in IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[31] Jiao S , Jin Z , Chang C , et al., Compression of Phase-Only Holograms with JPEG Standard and Deep Learning, *Applied Sciences*, 2018, 8(8).

[32] Kyoji Matsushima and Tomoyoshi Shimobaba, Band-Limited Angular Spectrum Method for Numerical Simulation of Free-Space Propagation in Far and Near Fields, *Opt. Express* 17, 19662-19673 (2009).

[33] Kaan Akşit, et al., kunguz/odak: Odak 0.1.6, *Zenodo*, jun 2021, doi:10.5281/zenodo.5035579

[34] Ping Su, et al., Fast Computer-Generated Hologram Generation Method for Three-Dimensional Point Cloud Model, *J. Display Technol.*12, 1688-1694 (2016).

[35] J.-S. Chen and D. P. Chu, Improved layer-based method for rapid hologram generation and real-time interactive holographic display applications, *Opt. Express*23, 18143-18155 (2015).

[36] Hao Zhang, Liangcai Cao, and Guofan Jin, Computer-generated hologram with occlusion effect using layer-based processing, *Appl. Opt.*56, F138-F143 (2017).

[37] Yan Zhao, et al., Accurate calculation of computer-generated holograms using angular-spectrum layer-oriented method, *Opt. Express* 23, 25440-25449 (2015).

[38] Shi, L., Li, B., Kim, C. et al. , Towards real-time photorealistic 3D holography with deep neural networks, *Nature*591, 234–239 (2021).

[39] Lee C , Diverdi S , Hollerer T, Depth-Fused 3D Imagery on an Immaterial Display, *IEEE transactions on visualization and computer graphics*, 2009, 15(1):p.20-33.

[40] Sheng Liu and Hong Hua, A systematic method for designing depth-fused multifocal plane, three-dimensional display *Opt. Express*18, 11562-11573 (2010)

[41] R. W. Floyd and L. Steinberg, An Adaptive Algorithm for Spatial Grayscale, *Proceedings of the Society of Information Display*, Vol. 17, No. 2, 1976, pp. 75-77.

[42] P.W.M. Tsang and T. -C. Poon, Novel method for converting digital Fresnel hologram to phase-only hologram based on bidirectional error diffusion, *Opt. Express 21*, 23680-23686 (2013)

[43] Christopher A. Metzler et al., prDeep: Robust Phase Retrieval with a Flexible Deep Network *arXiv:1803.00212*, 2018.

[44] Rivenson, Y., Zhang, Y., Günaydın, H. et al. , Phase recovery and holographic image reconstruction using deep learning in neural networks, *Light Sci Appl* 7, 17141 (2018). https://doi.org/10.1038/lsa.2017.141.

[45] Zhenbo Ren, Zhimin Xu, Edmund Y. Lam, Autofocusing in digital holography using deep learning, *Proc. SPIE 10499*, Three-Dimensional and Multidimensional Microscopy: Image Acquisition and Processing XXV, 104991V (23 February 2018).

[46] Zhenbo Ren, Zhimin Xu, and Edmund Y. Lam, Learning-based nonparametric autofocusing for digital holography, *Optica 5*, 337-344 (2018).

[47] T. Shimobaba, T. Kakue and T. Ito, Convolutional Neural Network-Based Regression for Depth Prediction in Digital Holography, *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, 2018, pp. 1323-1326, doi: 10.1109/ISIE.2018.8433651.

[48] Zhang, Yang and Wang, Hongru and Shan, Mingguang, Deep-Learning-Enhanced Digital Holographic Autofocus Imaging, *Association for Computing Machinery*, doi:10.1145/3408127.3408194, 2020.

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

[50] Md Zahangir Alom, et al., Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation, *arXiv:1802.06955*, cs.CV, 2018.

[51] Ozan Oktay , et al., Attention U-Net: Learning Where to Look for the Pancreas, *arXiv:1804.03999*, cs.CV, 2018.

[52] Wojciech Zaremba and Ilya Sutskever and Oriol Vinyals, Recurrent Neural Network Regularization, *arXiv:1409.2329*, cs.NE, 2015.

[53] Ming Liang and Xiaolin Hu, Recurrent convolutional neural network for object recognition, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3367-3375, doi: 10.1109/CVPR.2015.7298958.

[54] Kaiming He, et al., Deep Residual Learning for Image Recognition, *arXiv:1512.03385*, cs.CV, 2015.

[55] Ashish Vaswani and Noam Shazeer et al., Attention Is All You Need, *arXiv:1706.03762*, cs.CL, 2017.

[56] Chang, Angel X., et al., ShapeNet: An Information-Rich 3D Model Repository, *arXiv:1512.03012*, cs.GR, Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[57] Diederik P. Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimization, *arXiv:1412.6980*, cs.LG, 2017.

[58] Kaiming He, et al., Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, *arXiv:2015*, cs.CV, 2015.

[59] Paszke, Adam, et al., PyTorch: An Imperative Style, High-Performance Deep Learning Library, *Advances in Neural Information Processing Systems 32*, 8024–8035, Curran Associates, Inc..

[60] Born, Max, and Emil Wolf., Principles of optics: electromagnetic theory of propagation, interference and diffraction of light, *Elsevier*, 2013.

[61] Lequan Yu, et al., PU-Net: Point Cloud Upsampling Network, *arXiv:1801.06761*, cs.CV, 2018.

[62] Peng, Yifan, et al., Neural holography with camera-in-the-loop training, *ACM Transactions on Graphics.* 39. 1-14. 10.1145/3414685.3417802., 2020.

Appendix A

Source Code

Related source code is available in the following public GitHub repository:
`https://github.com/Wh1t3zZg/LDZX7-source-code`