The University of Hong Kong
Faculty of Engineering
Department of Computer Science

COMP4801 Final Year Project

3D reconstruction with single/multi-view images

Final Report

Title:	3D reconstruction with single/multi-view images
Supervisor:	Zhao Hengshuang
Student:	Jiang Zeyu (3035639056)
Date of Submission:	April 25, 2024

Acknowledgement

Sincere thanks to the supervisor Prof. Zhao Hengshuang for his guidance and support throughout the project. Also, I would like to thank my friends and family for their encouragement and support along the way.

Abstract

3D reconstruction from images is an important and challenging problem in computer vision. In this project, we proposed a two-stage framework to reconstruct 3D models of objects from sparse view images. We trained a multi-view diffusion model to predict the multi-view color and depth images of the object and then used an SDF-based model to reconstruct the 3D shape of the object. We also explore the possibility of using Gaussian Splatting [7] as the reconstruction module. We evaluated our diffusion model on the Google Scanned Object dataset [3] and the results show that our diffusion model can achieve competitive performance. Though our reconstruction module is not as good as the State-of-the-Art methods, we still propose in the future to improve the reconstruction module and the training process to achieve better results.

Contents

List of Figures

List of Tables

List of Abbreviations

DDPM	Denoising Diffusion Probabilistic Models
MLP	Multi Layer Perceptron
MVS	Multi-view Stereo
NeRF	Neural Radiance Field
VAE	Variational Autoencoders
SSIM	Structural Similarity Index, higher the better
PSNR	Peak Signal-to-Noise Ratio, higher the better
LPIPS	Learned Perceptual Image Patch Similarity, lower the better
FPS	Frames Per Second, higher the better
3DGS	3D Gaussians Splatting

1 Introduction

1.1 Background

Humans can effortlessly interpret the 3D structure of an object from a mere collection of 2D images. This marvelous ability rooted in our cognitive faculties results from imagination and strong prior knowledge from our visual experiences. However, enabling machines to perform 3D reconstructions just like humans remains an open challenge, while there are many substantial applications across medicine, games, augmented reality and more.

1.2 Motivation

In 3D reconstruction, classic multi-view stereo (MVS) [25, 26] methods can reconstruct 3D models from multiple images taken from known viewpoints. However, they struggle with textureless surfaces, lighting variations, and thin structures. MVS also requires many input views captured in a controlled setting.

Lately, numerous methods have adopted neural radiance field (NeRF) to model 3D scenes, demonstrating strong reconstruction performance and high fidelity results [16, 28, 17, 10]. They use implicit neural representations, directly leveraging the learnable neural networks, despite traditional voxel, mesh or point cloud methods. However, they normally require considerable input images with known poses and may produce blurry output when facing unseen areas.

More recently, 3D Gaussian Splatting achieves real-time rendering of a trained 3D scene using 3D Gaussians as an explicit representation [7]. It still adopts the learning techniques as in NeRF [16], allowing faster backpropagation and reconstruction of a 3D scene, though it still needs enough images with pose inputs.

On the track of image generation, denoising diffusion probabilistic models (DDPM) [6, 21] have shown remarkable performances. Techniques based on DDPM [6] primarily learn a noise predictor from the forward process, which continuously adds noise to natural images. This learned noise predictor is then used in the reverse process to generate images from the Gaussian noise, through a series of diffusion steps.

Thus, this project hypothesizes that the confluence of NeRF [16], 3D Gaussians Splatting [7] and diffusion models may represent a new paradigm in 3D reconstructions, especially under the sparse-views condition. Furthermore, reconstructing 3D models from merely a few images taken from your phone may have vast applications in various domains like gaming and augmented reality.

1.3 Project objectives

The key objectives of this project are to develop a framework for 3D reconstruction with sparse input images, which should be capable of:

- **Single-view 3D reconstruction:** This framework should be capable of reconstructing 3D models from single image inputs and synthesizing consistent hidden views using the prior knowledge from diffusion models.
- **Incremental multi-view enhancement:** This framework should be capable of refining the quality of 3D models generated from single-view inputs. Enhancements may include correcting the textures and geometry of the model with additional inputs.
- **Flexable input handling:** This framework should be capable of directly handling inputs from the open world without needing categorical priors, masks, or predefined poses. This flexibility will make the framework more adaptable to real-world, uncontrolled scenarios.
- **Evaluation and analysis:** Evaluate the reconstruction quality compared to SOTA methods in closed-world benchmarks quantitively and qualitatively for open-world inputs. Analyze tradeoffs between single vs multi-view reconstruction in terms of quality, reconstruction time, and other available metrics.

1.4 Outline of the report

The remaining parts of this report proceed as follows. Section 2 analyzes the current research state and identifies literature review gaps. Section 3 offers a discussion on methodology and describes the framework's construction. Section 4 presents the work that has been accomplished, what remains to be done, plans for the future, and problems encountered. We round off with a conclusion restating objectives and progress.

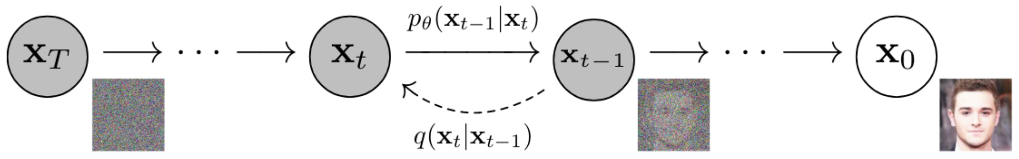
2 Literature Review

The literature review summarizes related current research. We will first go through the basics of diffusion models, nerual radiance field [16] and 3D Gaussians Splatting [7]. Afterwards, we'll look into some view-conditioned diffusion models that are able to generate images from novel viewpoints leveraging the prior knowledge of large pre-trained 2D diffusion models. Finally, we'll see how diffusion models reforms the 3D generation and reconstruction tasks.

2.1 Diffusion Models

2.1.1 Denoising Diffusion Probabilistic Model

The denoising diffusion probabilistic model (DDPM) [6] marks a new paradigm for image generation. The orginal DDPM [6] models a Markov chain as shown in the figure below.



$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 1: The Markov chain of DDPM [6]

Given $\mathbf{x}_0$ d representing the observed distribution of an arbitrary natural image and $\mathbf{x}_T$ representing the pure Gaussian noise, DDPM [6] learns a noise predictor from the forward process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ of adding noise from $\mathbf{x}_0$ to $\mathbf{x}_T$. Since we are modeling a Markovian process, the forward process can be modeled as:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{1}$$

Also, DDPM [6] assumes all latent variables in the encoder are a Gaussian distribution centered around the previous one and sets the mean and variance of the Gaussian encoder as follows, with α_t as a coefficient that may vary.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I}) \tag{2}$$

With α_t evolving over steps t and $p(\mathbf{x}_T)$ being a standard Gaussian distribution, i.e. $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$:

$$p(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{3}$$

DDPM [6] then defines its trainable noise predictor by minimizing the KL divergence. ϵ is a random variable sampled from standard Gaussian. The final loss function used to train the noise predictor evolves to:

$$L_\theta = \mathbb{E}_{t,\S+0,\epsilon}\left[\left|\left|\epsilon - \epsilon_\theta\left(\sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, t\right)\right|\right|^2\right] \tag{4}$$

2.1.2 Latent Diffusion Models

Due to the sequential and repeated nature of the DDPM [6], training and inference of the model should be performed step by step. Meanwhile, the DDPM [6] directly operates on the pixel space, it requires a large amount of memory during training and generating high resolution images. To resolve the above issues, the latent diffusion model [21] is trained to generate the latent representations of images, which applies the denoising process in the latent space. It utilizes a variational autoencoder (VAE) to encode the image into latent space during training while decoding the latent representations into images during inference process.

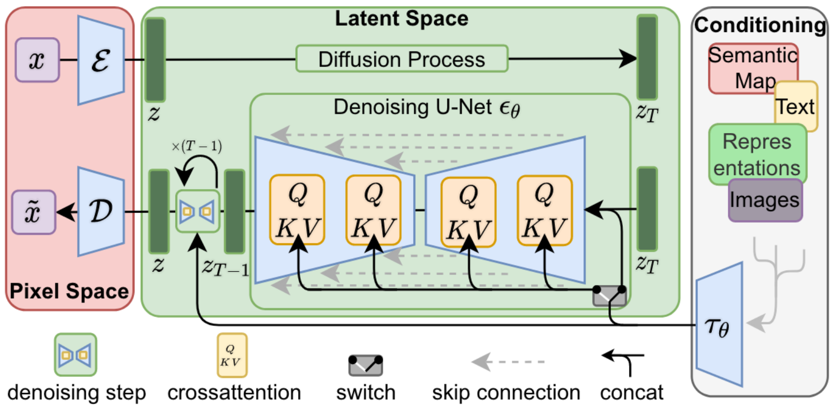


Figure 2: Architecture of latent diffusion model [21]

In the latent space, a U-Net [22] structure is used to predict the noise. This U-Net consists an encoder and a decoder. While the encoder downsamples the image's latent representation, the decoder upsamples it back to the original size with less noise, making this U-Net output predicting the noise residual which is used to denoise the image's latent representation. Moreover, this model is also capable of conditioning on the additional input, such as text, image or semantic map via a cross-attention layer.

The cross-attention layer works with a pre-trained embedding model. Taking Stable Diffusion [21] as an example. It utilizes CLIP [19] to encode text prompt into a text embedding vector that could be fed into the cross attention layer. This makes it possible for the image generation processes to focus on the input text prompts.

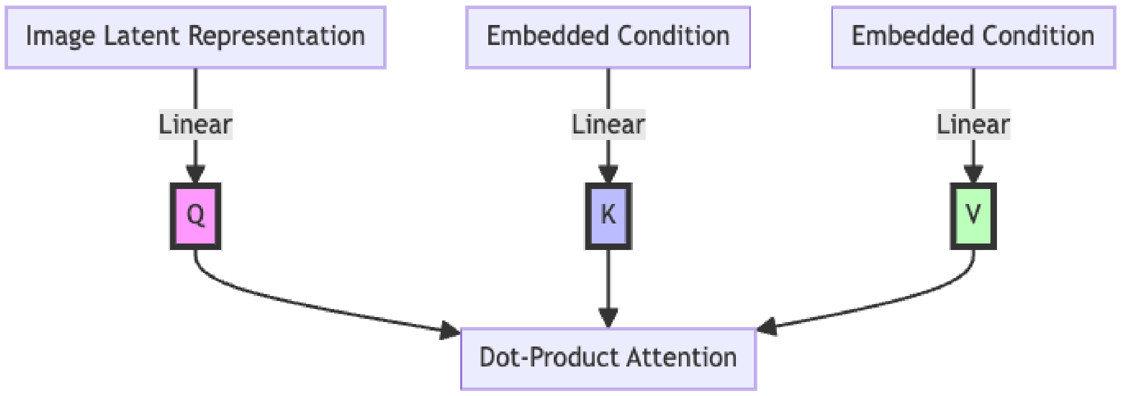


Figure 3: Cross Attention Mechanism

2.2 Neural Radiance Field

In the vanilla NeRF [16], a scene could be represented using a function taken in coordinate $\mathbf{x} = (x, y, z)$ in the 3D space, along with a viewpoint $\mathbf{d} = (\theta, \phi)$ to reconstruct colors $\mathbf{c} = (r, g, b)$ and densities σ along the ray.

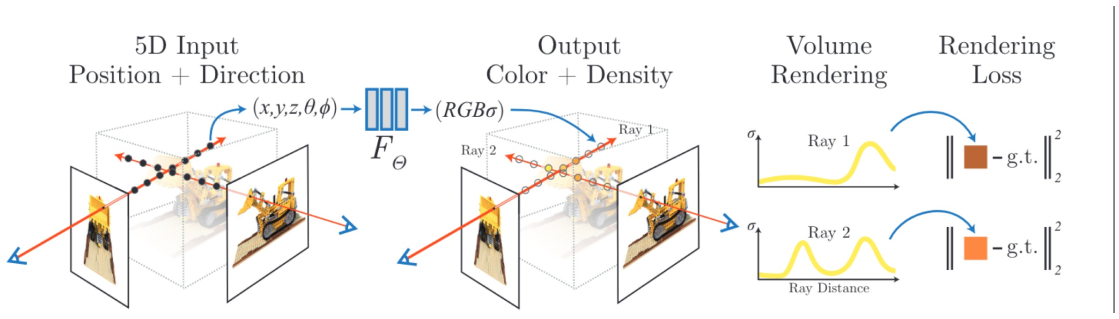


Figure 4: Neural Radiance Field [16] scene representation

The vanilla design of NeRF [16] uses sets of MLPs to model the radiance and density functions with trainable parameters. These MLPs are then trained using multi-view images of a scene to learn the implicit neural representation. Afterward, the novel view can be synthesized by querying these MLPs and rendering using volume renderings. Given the camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, the color in the bounds t_n to t_f can be derived as [16]:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{5}$$

In addition, the author found that directly operating on coordinate $\mathbf{x} = (x, y, z)$ and viewpoint $\mathbf{d} = (\theta, \phi)$ results in underfitting problem, that the network cannot perform well when there's high variation of color and geometry in the scene. Therefore, it uses a positional encoding to embed the 5D vector into a higher dimensional space utilizing the sine and cosine functions, which is similar to that in the Transformer [27] architecture. Meanwhile, it introduces the volume rendering method with hierarchical volume sampling to avoid useless repeated sampling.

However, under our scenario of 3D reconstruction with sparse input images, the original design of NeRF [16] often produces blurry outputs, since itself does not have the ability to make prediction on the unseen areas. Meanwhile, due to its implicit representation design, it's slow to train and render. These downsides have been emphasized in future works. Instant-NGP [17] used a multi-resolution hash encoding, prominently improves the speed in training and rendering. NeuS [28] and Neuralangelo [10] used implicit signed distance function and novel volume rendering method to reconstruct smooth surface. Mip-NeRF 360 [1] focuses on resolve the blurry boundary issue by using online distillation and distortion-based regularizers.

2.3 3D Gaussians Splatting

3D Gaussians Splatting [7] is a more recent study adopting the key idea of training in NeRF [16] while using a relatively explicit 3D scene representation. It outperforms most NeRF-based method in terms of quality (SSIM, PNSR) and efficiency (FPS, Training Time).

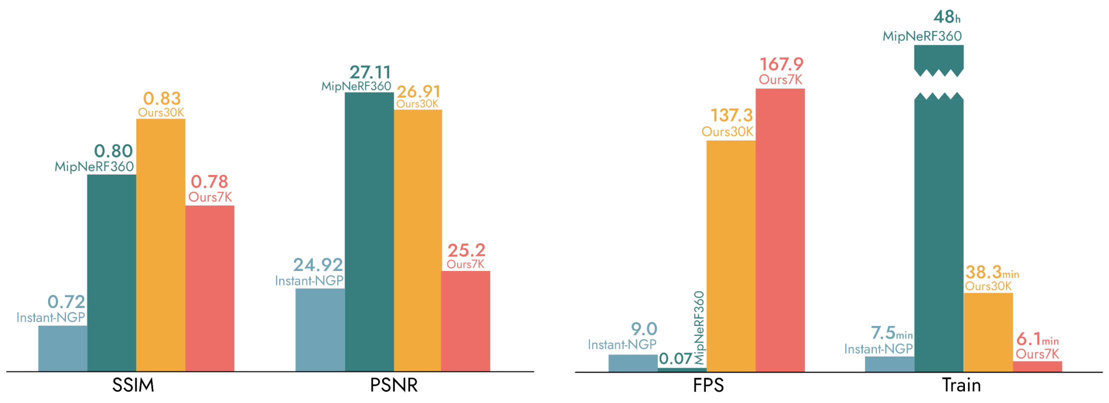


Figure 5: Performance Evaluation of 3D Gaussians Splatting [7]

3D Gaussians Splatting [7] uses anistropic 3D Gaussians as an explicit representation of a 3D scene. Each Gaussian can carry the feature vector like opacity and spherical harmonics to fit the directional appearance of the radiance field and itself carries position information naturally by centering itself at a specifc position. Then, by projecting relavant Gaussians onto a 2D plane using a differentiable tile rasterizer, it can render the image from a specific camera viewpoint. Compared to Point-NeRF [29], which also stores features in points but using volume rendering and linear interpolation, this process effectively leverages the explicit representation of the point cloud, the scene could be rendered way more faster than some NeRF-based methods which need to make inference through multiple MLPs.

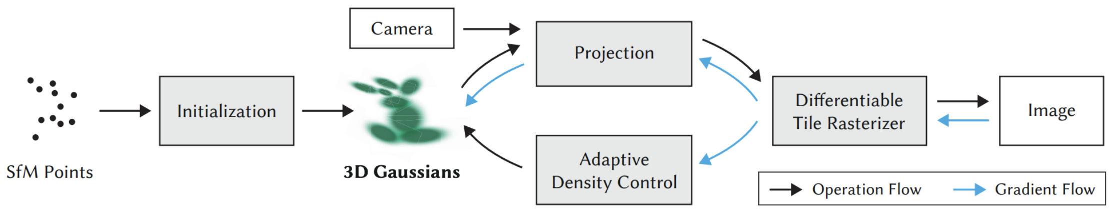


Figure 6: Flow of 3D Gaussians Splatting [7]

Another major contribution of the 3D Gaussian Splatting is its adaptive density control method. The control method basically checks on the gradient applies to each Gaussian. When the gradient is too large, proving that one Gaussian may not be enough to represent the scene at its position, then this Gaussian will be cloned or splitted as in the following diagram. Similarly, after some interations, remove the Gaussians that have opacity below the threshold, which means these Gaussians have nearly no impact on the rendering.

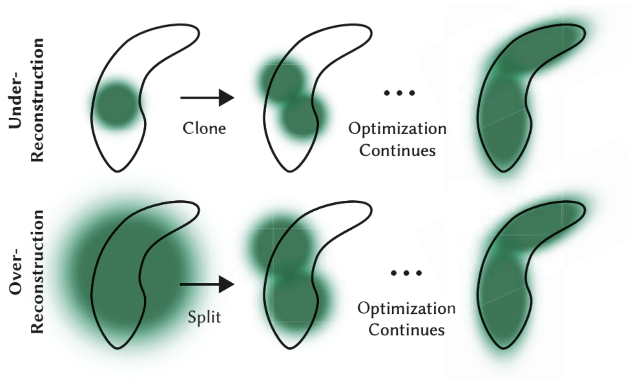


Figure 7: Densify (Clone/Split) Scheme of 3D Gaussians Splatting [7]

However, same as NeRF [16] methods, 3D Gaussian splatting still requires a large amount of images input, with camera poses. It will produce blurry outputs for unseen areas as well due to lack of prediction ability. Meanwhile, we found that 3D Gaussians Splatting cannot produce a sharp output when there's a large variance in shape and color, or in the boundary position.

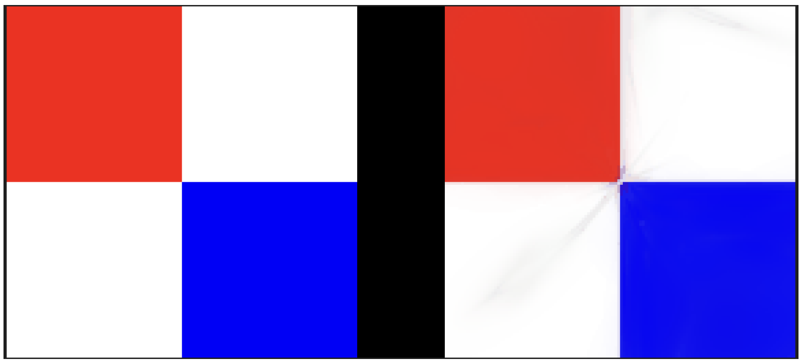


Figure 8: Simulation on recovering the left-side ground truth using 3DGS [7]

2.4 View-conditioned diffusion models

The pioneer work Zero-1-to-3 [12] make use of the large pre-trained 2D diffusion models, Stable Diffusion [21], to learn a control mechanism that could manipulate camera viewpoint during the image generation process, enabling the zero-shot ability.

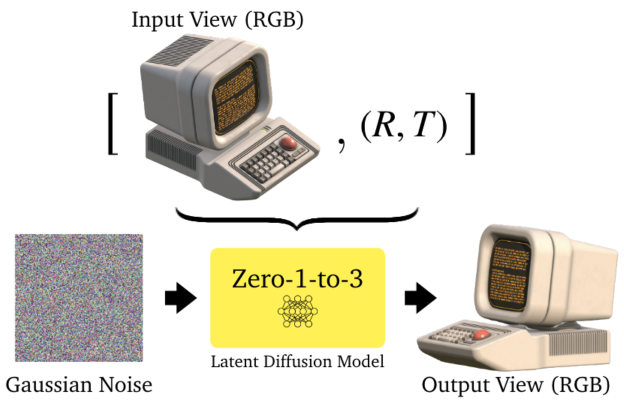


Figure 9: Model of Zero-1-to-3 [12]

It uses the Objaverse [2] dataset, which contains a large amount of 3D objects, processing it into viewpoint and image pairs to finetune the Stable Diffusion [21] model. However, since it's generating only one image at a time, even using the same view prompt, due to the probabilistic nature of the diffusion model, this may lead to inconsistency problems. One is the multi-face problem that the diffusion model repeatedly generates content that might be invisible in some angle, and the other is the content drift problem that some content in the image might be loss or gradually become other things.

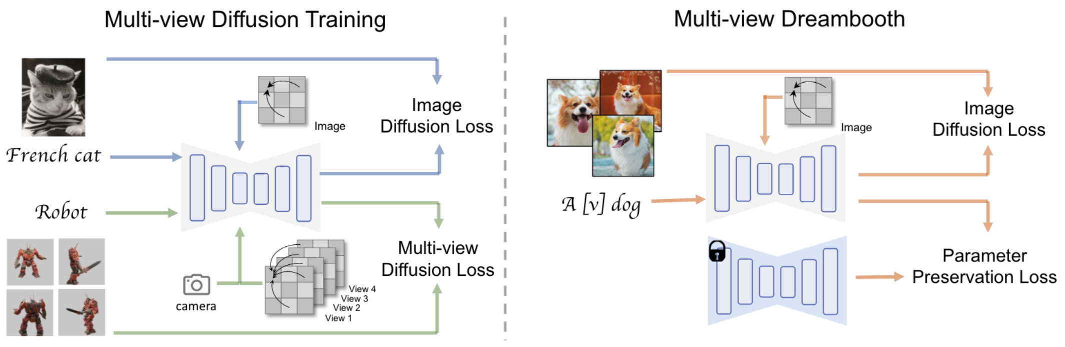


Figure 10: Pipeline of MVDream [24]

To resolve the above mentioned issue and keep the consistency cross views, SyncDreamer [13] and MVDream [24] both propose to generate multiple views at the same time. While SyncDreamer [13] finetunes from Zero-1-to-3 [12] and aims to model a joint distribution across different views, using a synced noise predictor for views from different angles. MVDream [24] finetunes from pre-trained 2D diffusion models and uses a 3D attention mechanism across the views to maintain consistency. Future work Wonder3D

[14] also utilizes this 3D self-attention mechanism to ensure multi-view consistency, as well as generating paired normal images for 3D reconstruction using SDF method.

2.5 Diffusion Guided 3D Generation and Reconstruction

DreamFusion [18], as a pioneer work, introduces the score distillation sampling technique that allows the 3D reconstruction process to be guided by the pre-trained diffusion models, as its pipeline shown in the following figure.

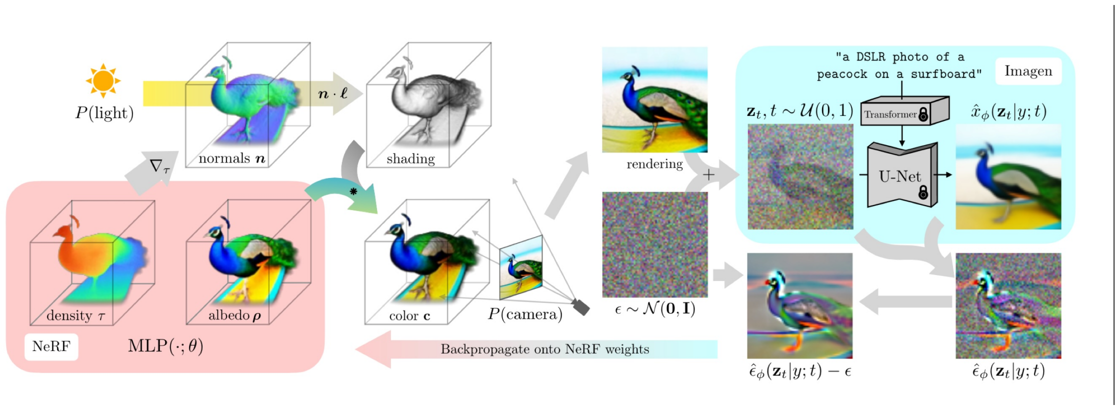

P(light)
normals $\boldsymbol{n}$
$\boldsymbol{n}\cdot\boldsymbol{\ell}$
shading
∇_τ
density τ
albedo $\boldsymbol{\rho}$
NeRF
MLP$(\cdot;\theta)$
color $\mathbf{c}$
P(camera)
rendering
$\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})$
Backpropagate onto NeRF weights
"a DSLR photo of a peacock on a surfboard"
Imagen
$\mathbf{z}_t, t\sim\mathcal{U}(0,1)$
Transformer
U-Net
$\hat{x}_\phi(\mathbf{z}_t|y;t)$
$\hat{\epsilon}_\phi(\mathbf{z}_t|y;t)-\epsilon$
$\hat{\epsilon}_\phi(\mathbf{z}_t|y;t)$

Figure 11: Pipeline of DreamFusion [18]

DreamFusion [18] firstly initialize the NeRF [16] model with parameters θ. Then, given a specific camera viewpoint, it uses the NeRF [16] model to render the 2D image and adds noise to this rendered image. Afterwards, using the text prompt describing the viewpoint and the object that should be generated and the noised rendered image as the input to the pre-trained diffusion model. The diffusion model should be able predict the noise we just added to the rendered image. The difference between this prediction and the noise we add will be backpropagate onto the weights of the NeRF [16] model. Under this setting, DreamFusion proposes the following SDS loss, which is derived from the loss of the diffusion model and let the gradient to flow outside:

$$\nabla_\theta \mathcal{L}_{SDS}(\phi, \mathbf{x}=g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t)(\hat{\epsilon}_\phi(\mathbf{z}_t; y, t)-\epsilon)\frac{\partial \mathbf{x}}{\partial \theta}\right] \tag{6}$$

However, the drawbacks of this method is obvious. Firstly, the vanilla DreamFusion directly uses a text-to-image diffusion model, Imagen [23], to serve as the guidance. As the training procedure of these kind of models did not focus on the view angle information, directly using text prompts to control the angle may be hard. Though this can be resolved by using fine-tuned view-conditioned diffusion model like Zero-1-to-3 [12], its need of per instance optimization still leads to ineffiency. Problem of inaccurate colors and textures also happens from time to time.

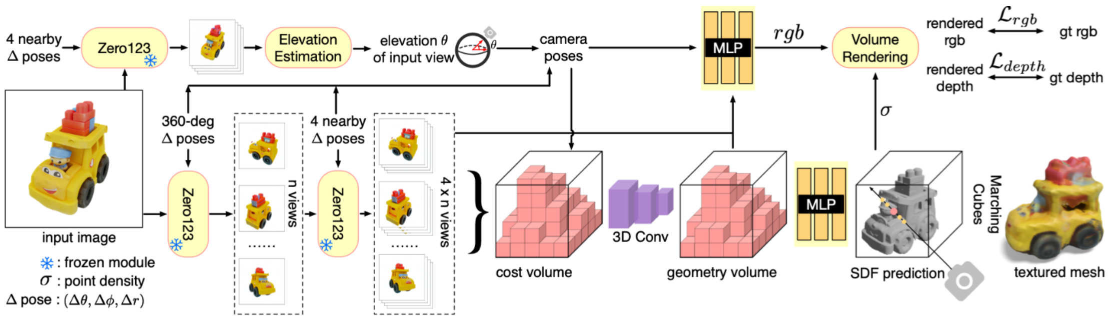


Figure 12: Pipeline of One-2-3-45 [11]

Under this scenario, One-2-3-45 [11] directly utilizes the Zero-1-to-3 [12] to generate novel views from multiple angles and reconstruct them using the SDF-based method NeuS [28]. Though it could generate the mesh in one forward pass, it still suffers from the inconsistency problem from Zero-1-to-3. Also, it needs to retrain the 3D datasets for the latter part of depth prediction using the SDF method, which may affect the generalization ability.

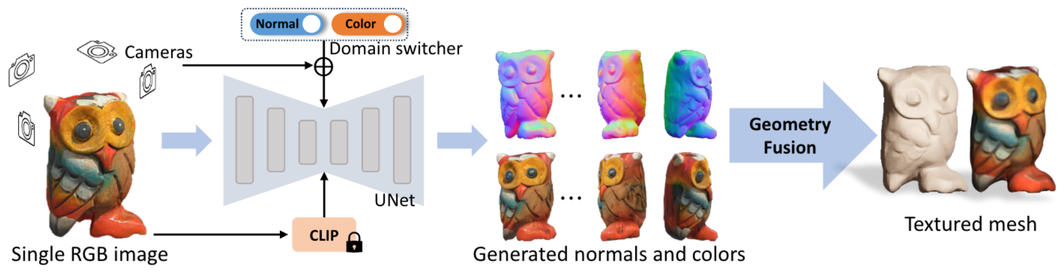


Figure 13: Pipeline of Wonder3D [14]

Wonder3D [14], a more recent work, directly fine-tunes the diffusion model to be able to output the paired normal images. In that case, there's no need to estimate depth information from the colored image, and this output could be directly fed into the SDF-based reconstruction method, achieving a smooth surface while maintaining the efficiency by reconstruction in one forward pass as well.

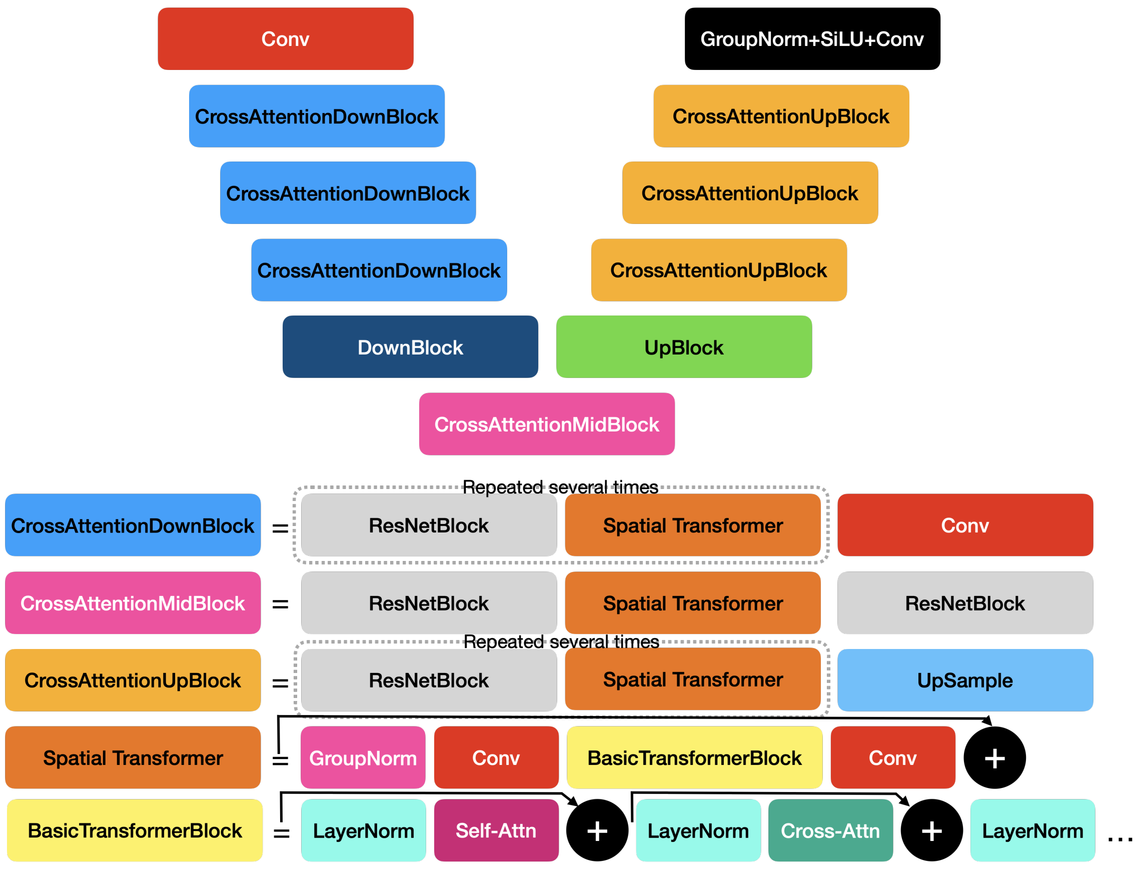
3 Methodology

The project will be divided into two parts, a multi-view diffusion models that generate paired color and depth images of the object from multiple sides, and a reconstruction model (signed-distance function (SDF) based and Gaussian splatting based) that takes the generated images and reconstructs the 3D object. The detailed methodology for each part is described below.

3.1 Multi-view color-depth diffusion models

In developing the multi-view color-depth diffusion models, we build upon the structure of the Stable Diffusion model [21]. Our approach involves modifying the original U-Net architecture and incorporating embeddings for camera positions and the color/depth task.

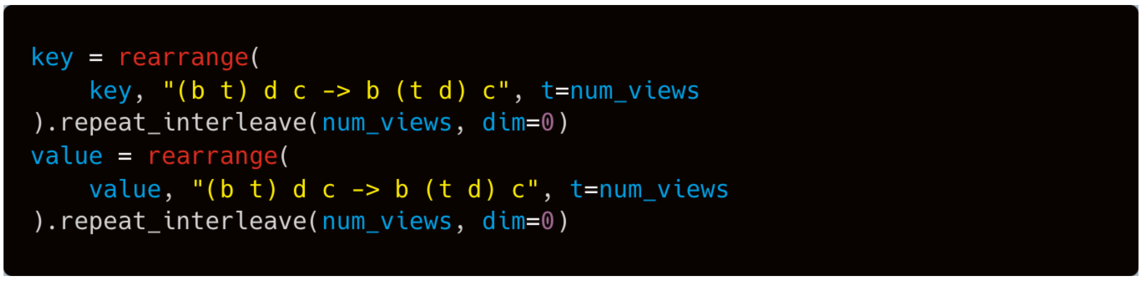
3.1.1 Modifications on the original U-Net



Conv
GroupNorm+SiLU+Conv
CrossAttentionDownBlock
CrossAttentionUpBlock
CrossAttentionDownBlock
CrossAttentionUpBlock
CrossAttentionDownBlock
CrossAttentionUpBlock
DownBlock
UpBlock
CrossAttentionMidBlock
Repeated several times
CrossAttentionDownBlock = ResNetBlock Spatial Transformer Conv
CrossAttentionMidBlock = ResNetBlock Spatial Transformer ResNetBlock
Repeated several times
CrossAttentionUpBlock = ResNetBlock Spatial Transformer UpSample
Spatial Transformer = GroupNorm Conv BasicTransformerBlock Conv +
BasicTransformerBlock = LayerNorm Self-Attn + LayerNorm Cross-Attn + LayerNorm ...

Figure 14: Basic decomposition of blocks inside the U-Net of Stable Diffusion [21]

Figure 14 illustrates the basic decomposition of blocks within the U-Net of Stable Diffusion [21], with a primary focus on the attention blocks. As our objective is to generate multiple images from designated perspectives of the object, it is necessary to maintain consistency among these images. To achieve this, we adapt the self-attention block into a multi-view attention block, as described in MVDream [24] to build up the global dependency for each view.



```
key = rearrange(
    key, "(b t) d c -> b (t d) c", t=num_views
).repeat_interleave(num_views, dim=0)
value = rearrange(
    value, "(b t) d c -> b (t d) c", t=num_views
).repeat_interleave(num_views, dim=0)
```

Figure 15: Build multi-view attention by leveraging the batch dimension

We realize this adaptation by leveraging the existing batch dimension to manage multiple views as in figure 15. Since we want to build a six-view diffusion model, we directly repeat the input view six times to serve as the input to the model, then we could again copy the information from the other views to the current view when calculating the key and value in attentions. This way, the model can learn the global dependency among all views. Meanwhile, this modification won't affect the original U-Net structure, allowing us to better leverage the pre-trained weights from the Stable Diffusion model.

Afterwards, since we'll also need to generate depth images and each depth image should be paired with the corresponding color image. Thus, we add another cross-attention block right after our multi-view attention block to build the dependency between the generated color and depth images. However, unlike the cross-domain attention introduced in Wonder3D [14], we only enable this kind of attention in the bottleneck block (CrossAttentionMidBlock) of the U-Net and the last CrossAttentionUpBlock for efficiency. Also, there's no need for depth images to occupy three channels as the color images or normal images, we implement a strategy that only uses one channel for depth images and broadcast results to other channels when needed. This way, we can largely reduce the VRAM usage during training and inference.

3.1.2 Embeddings for camera positions and tasks

In Zero-1-to-3 [12], MVDream [24], and SyncDreamer [13], besides the standard embeddings for the camera (RT matrix), an embedding for the elevation angle is also included. This angle is defined as the angle between the camera and the object plane. Zero-1-to-3 [12] incorporates this elevation angle to better control the camera position, whereas MVDream [24] uses it primarily because they preprocess the data with a random elevation angle. However, requiring users to estimate the elevation angle during inference is impractical and can lead to suboptimal results due to inaccurate estimations. Therefore, we have decided to omit the elevation angle embedding by predefining fixed camera positions and rotating the entire camera plane rather than merely elevating the camera. This approach maintains the robustness of the model.

For embeddings of the camera, we use a two-layer MLP followed by a sin-cos embedding just like the position encoding in Transformer [27]. The input to the MLP is the camera position matrix in the world coordinate system, and the output is directly concatenated with the timestep embedding.

Since we also need to control the model to generate color or depth images, unlike Wonder3D [14] which concatenates the task embedding, which is a one-hot vector, with the input camera position matrix, we directly append the task embedding to the timestep embedding, which we found that it's easier for the model to learn the task information.

3.1.3 Diffusion losses and EMA

In the original Stable Diffusion [21], and a lot of other latent diffusion models, the loss function is derived as the MSE loss between noise added to the latent and the noise predicted by the U-Net. However, there are many complaints about the convergence speed. To address this issue, we followed the SNR loss described in [5] and applied the exponential moving average (EMA) to the U-Net.

3.2 SDF-based reconstruction model

NeuS [28] introduces the idea of using a signed-distance function (SDF) to represent the surface of a 3D object. The SDF is a function that returns the shortest distance from a point to the surface of the object. This method could be integrated well with our generated depth images. However, due to our usage of orthogonal cameras and rotation of the camera plane, modifications are still needed in the process of rays generation. Meanwhile, we also incorporated the hash grid encoding in InstantNGP [17] and numerical gradient in NeuralAngelo [10] for better efficiency and surface. Details are described below.

3.2.1 Normal map calculation

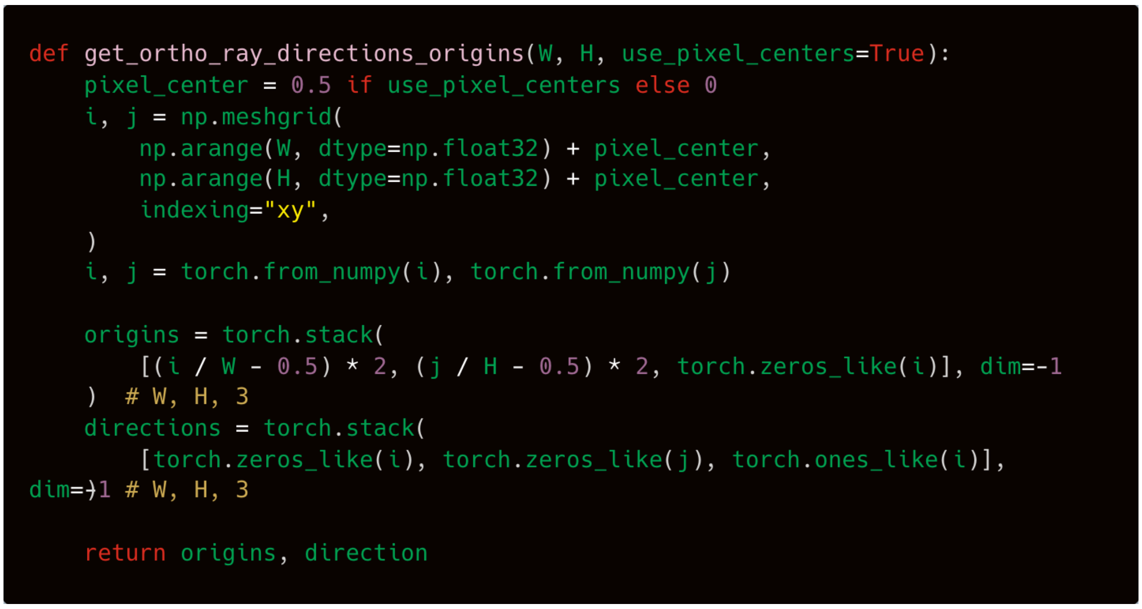
Our diffusion model can only produce estimated color and depth images. While most NeRF-based reconstruction methods are designed for images taken in the real world, they may not handle the errors in the generated images well. In that case, we adapt the Geo-Aware loss from Wonder3D [14], which is built upon normal maps. Thus, we calculate the normal maps from the depth images in the camera view by directly calculating the gradients of the depth images. However, since the depth images are generated and clipped, noises hugely affect the calculated normal map. To address this issue, we apply a Gaussian filter directly to the unclipped depth tensor and then calculate the gradients. This way, we can largely reduce the noises in the normal map.

```python
def depth2normal(depth_map: np.ndarray):
    rows, cols = depth_map.shape
    depth_map = gaussian_filter(depth_map.astype(float), sigma=1.0)

    x, y = np.meshgrid(np.arange(cols), np.arange(rows))
    x = x.astype(np.float32)
    y = y.astype(np.float32)

    # Level and strength
    dz = 2.0

    # Calculate the partial derivatives of depth with respect to x and y
    dx, dy = np.gradient(depth_map)
    dx *= dz
    dy *= dz

    # Compute the normal vector for each pixel
    normal = np.dstack((dy, -dx, np.ones((rows, cols))))
    norm = np.sqrt(np.sum(normal**2, axis=2, keepdims=True))
    normal = np.divide(normal, norm, out=np.zeros_like(normal), where=norm !=
0)
    # Map the normal vectors to the [0, 255] range and convert to uint8
    normal = (normal + 1) * 127.5
    normal = normal.clip(0, 255).astype(np.uint8)

    return normal
```

Figure 16: Code for normal map calculation

3.2.2 Orthogonal rays generation

In the original NeRF [16], the rays are generated by the camera position and the pixel position on the image plane. However, in our case, we need to generate rays that are orthogonal to the camera plane. To achieve this, we first generate rays in the camera coordinate system and make sure they are all parallel to the camera plane. Then, we convert them into the world coordinate system which aligns the design in NeRF [16]. Through this process, all the rays will be directly pointing to the object.



```
def get_ortho_ray_directions_origins(W, H, use_pixel_centers=True):
    pixel_center = 0.5 if use_pixel_centers else 0
    i, j = np.meshgrid(
        np.arange(W, dtype=np.float32) + pixel_center,
        np.arange(H, dtype=np.float32) + pixel_center,
        indexing="xy",
    )
    i, j = torch.from_numpy(i), torch.from_numpy(j)

    origins = torch.stack(
        [(i / W - 0.5) * 2, (j / H - 0.5) * 2, torch.zeros_like(i)], dim=-1
    )  # W, H, 3
    directions = torch.stack(
        [torch.zeros_like(i), torch.zeros_like(j), torch.ones_like(i)],
dim=-1 # W, H, 3

    return origins, direction
```

Figure 17: Code for orthogonal rays generation

3.2.3 Reconstruction losses

Our reconstruction loss function is defined by the equation:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{MSE-Depth}} + \lambda_2 \mathcal{L}_{\text{MSE-RGB}} + \lambda_3 \mathcal{L}_{\text{Mask}} + \lambda_4 \mathcal{R}_{\text{eikonal}} + \lambda_5 \mathcal{R}_{\text{Geo Aware}} \tag{7}$$

Here, $\mathcal{L}$MSE-Depth represents the mean squared error (MSE) loss between the predicted and ground truth depth. $\mathcal{L}$MSE-RGB denotes the MSE loss between the predicted RGB values and the ground truth RGB. $\mathcal{L}$Mask penalizes the model for predicting color and depth values outside the object boundaries. $\mathcal{R}$eikonal ensures the gradient of the signed distance function (SDF) maintains a unit value. $\mathcal{R}_{\text{Geo Aware}}$ applies the Geo-Aware loss as defined in Wonder3D [14].

Additionally, we employ a loss ranking mechanism that assigns varying weights to different generated views. This approach allows for the incorporation of real-captured images or depth data into the model with higher weights, enabling more accurate reconstruction based on real data as opposed to relying solely on estimates from the diffusion model.

3.2.4 Performance improvement

We further incorporated the model with some performance improvement techniques. For instance, we applied the hash grid encoding from InstantNGP [17] to accelerate the rendering process. We also utilized the numerical gradient and progressive hashing from NeuralAngelo [10] to improve the surface quality of the reconstructed object as well as the efficiency of the model.

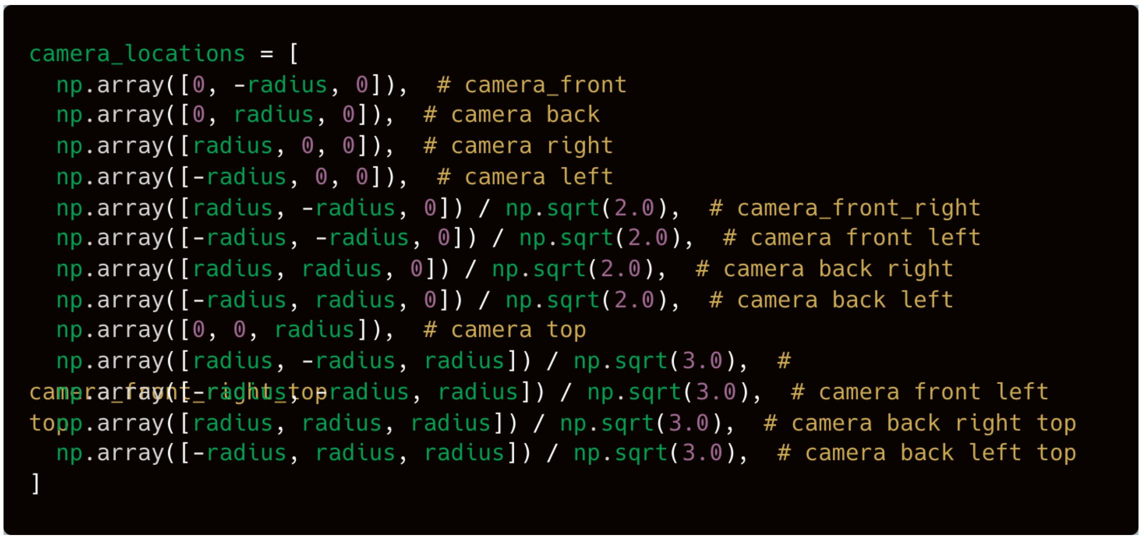
3.3 Gaussian Splatting-based reconstruction model

Gaussian Splatting [7], with its explicit representation of the 3D scene and more efficient rasterization process, significantly outperforms the NeRF-based implicit representation in terms of efficiency, thereby enabling real-time rendering. However, this method requires a sparse point cloud for initialization, normally from the SfM (Structure-from-Motion) method, which is not available in our case. Meanwhile, the performance can be severely affected by a point cloud initialized randomly. Moreover, the original Gaussian Splatting CUDA kernel does not support depth calculation. To address these shortcomings, we modified the CUDA kernel and the Python bindings to compute depth, defined as the distance to a Gaussian center where the opacity falls below a specified threshold. However, we still have difficulties with the point cloud initialization, and the integration of the Gaussian Splatting-based reconstruction model remained incomplete at the project's conclusion. We plan to continue refining this aspect in future work.

4 Experiments

4.1 Data preparation

To train our diffusion model to generate multi-view color and depth images given the single-view color image input, we prepare the training data from the Objaverse dataset [2]. This dataset contains over 800K+ annotated 3D objects with 46,832 objects having LVIS [4] labels. We then used this subset and rendered the objects from a total of 12 views: front, back, right, left, front-right, front-left, back-right, back-left, top, front-right-top, front-left-top, and back-right-top. The camera positions are generated by codes in figure 18.



```
camera_locations = [
  np.array([0, -radius, 0]),  # camera_front
  np.array([0, radius, 0]),  # camera back
  np.array([radius, 0, 0]),  # camera right
  np.array([-radius, 0, 0]),  # camera left
  np.array([radius, -radius, 0]) / np.sqrt(2.0),  # camera_front_right
  np.array([-radius, -radius, 0]) / np.sqrt(2.0),  # camera front left
  np.array([radius, radius, 0]) / np.sqrt(2.0),  # camera back right
  np.array([-radius, radius, 0]) / np.sqrt(2.0),  # camera back left
  np.array([0, 0, radius]),  # camera top
  np.array([radius, -radius, radius]) / np.sqrt(3.0),  #
camera_front_right_top
  np.array([-radius, -radius, radius]) / np.sqrt(3.0),  # camera front left
top
  np.array([radius, radius, radius]) / np.sqrt(3.0),  # camera back right top
  np.array([-radius, radius, radius]) / np.sqrt(3.0),  # camera back left top
]
```

Figure 18: Using numpy to generate camera positions for 12 views

Before rendering the object, we first normalize it to fit within a unit sphere. We then create a Blender pipeline in Python to render paired color and depth images at a resolution of 512x512 pixels. It should be noted that we applied a mapping node to map the depth values by deducting the distance between the camera and the sphere, thus normalizing the depth from the camera to the unit sphere where the object is placed to a range of [0, 1]. Though we rendered a total of 12 views, in practice, we only leveraged the 8 views parallel to the camera plane for the training process.

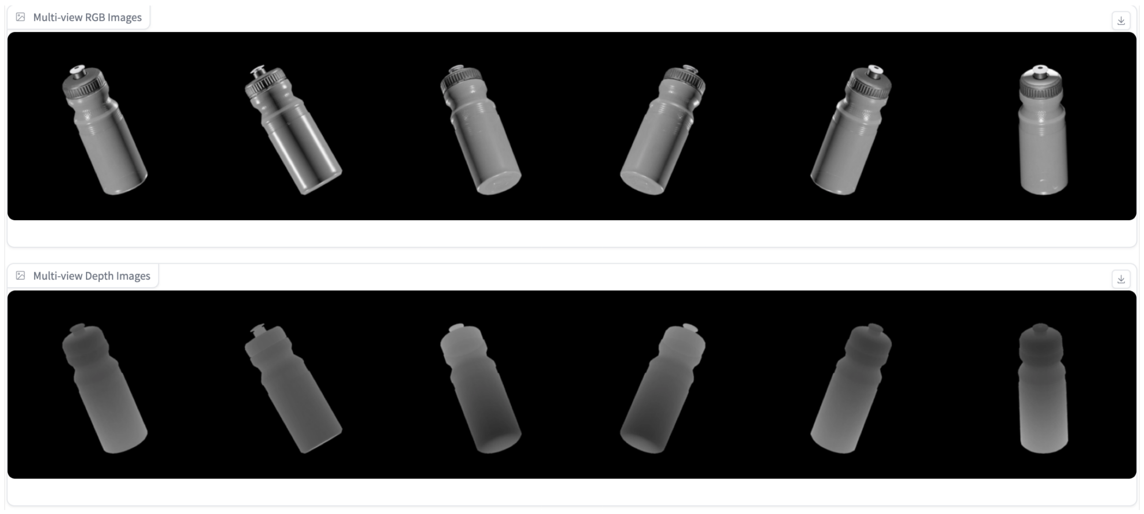


Figure 19: Sample of rendered color and depth images

4.2 Training of the multi-view diffusion model

4.2.1 Data preprocessing

Even after rendering the images, we must preprocess the data before feeding it into the model. We first resize the images to 256x256 pixels and normalize the pixel values to the range of [0, 1]. Since the original images are in RGBA format, we convert them to RGB by adding random backgrounds in black, white, and gray. We then divide the data into training, validation-train, and validation sets. The validation set contains the first 32 samples from the entire dataset, and the training set contains the remainder. The validation-train set, which includes the first 32 samples from the training dataset before shuffling, is used to monitor the training process and assess how well the model is learning. The random backgrounds are only used in the training set, while the other two sets use white backgrounds only. The mixed ground truth images in the validation train set for stage one training are shown in figure 20.

4.2.2 Two-stage training

When we initially added cross-attention between the color and depth images, the model struggled to converge as in figure 21. It was unable to learn the task and camera embeddings and could not generate images on a corresponding background based on the input. To address this issue, we removed the cross-attention in the first stage and trained the model to generate color and depth images randomly. Specifically, when pro-

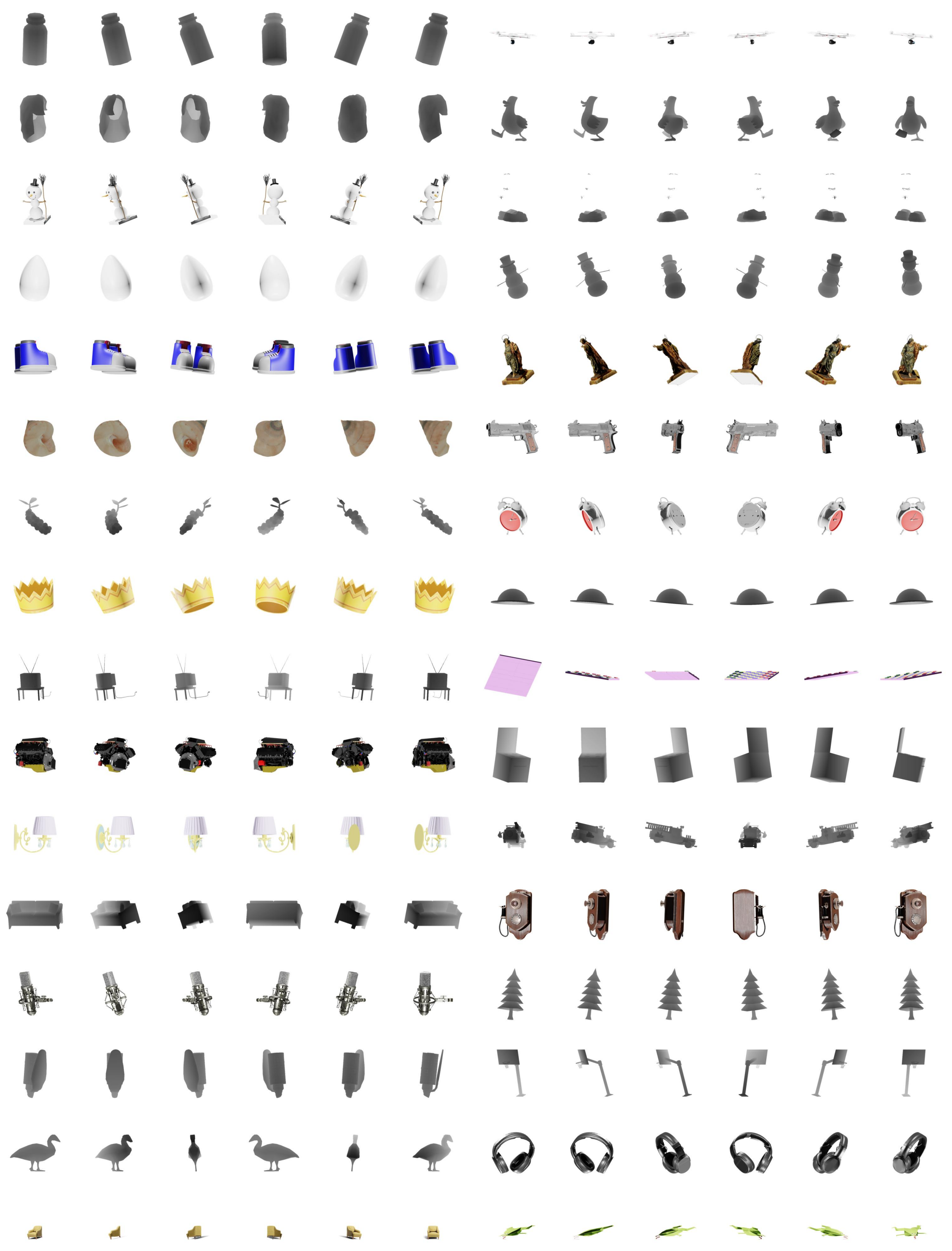


Figure 20: The mixed ground truth images in the validation train set for stage one training. The model is only given the color image of the front view and asked to generate the color/depth images of other views randomly

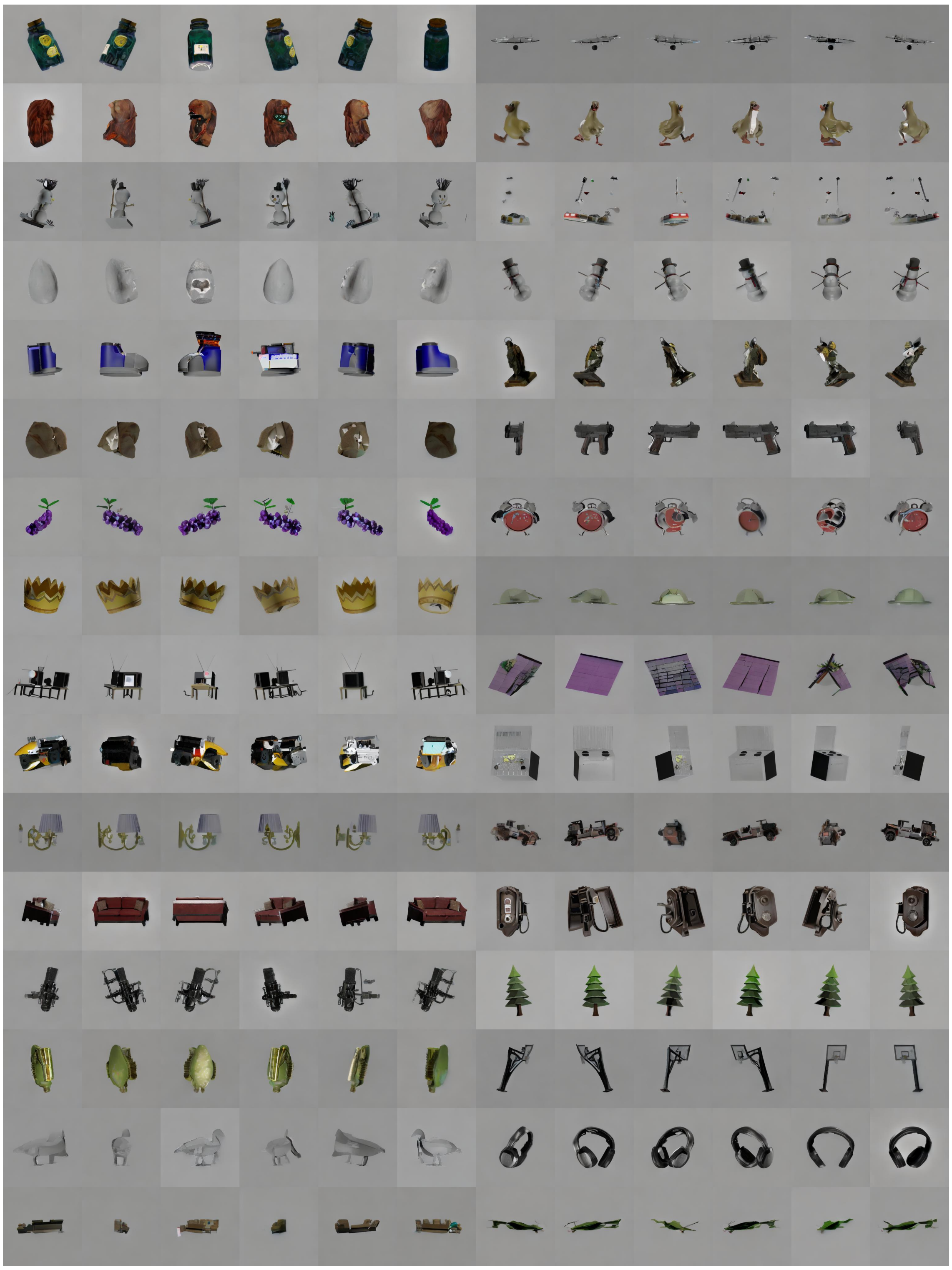


Figure 21: Slow convergence of diffusion model after 10k iterations on color images

When incorporating cross-attention between color and depth images, the model struggled to converge, showing mixed color backgrounds and random camera positions vided with the front color image, we randomly asked the model to generate multi-view color or depth images by changing the task embedding. In the second stage, we reintroduced cross-attention between color and depth images and trained the model to generate multi-view color and depth images given a single-view color image input. Since stage one essentially involves fine-tuning with nearly no new parameters added to the model, it converges much faster than the original setup. Only minimal further fine-tuning is required in stage two to enable the model to consistently generate paired multi-view color and depth images.

4.2.3 Pre-trained weights and hyperparameters

We initialized our model with pre-trained weights from the Stable Diffusion Image Variation[8] and fine-tuned it on our dataset. We employed the AdamW optimizer with a constant learning rate of 1e-4 for stage one and 5e-5 for stage two, with warm-up steps set at 100. Due to VRAM constraints, the batch size was limited to 8; however, we utilized the gradient accumulation technique to accumulate 16 steps, effectively simulating a batch size of 128. The model underwent training for 20,000 update steps in stage one and 5,000 update steps in stage two. The first stage took approximately 4 days on a single NVIDIA A100 40G PCIE GPU, and the second stage took another 2 days. We also discovered that the pre-training weights were crucial, and even without training, the model could generate images that closely resembled the input image as shown in figure 22.

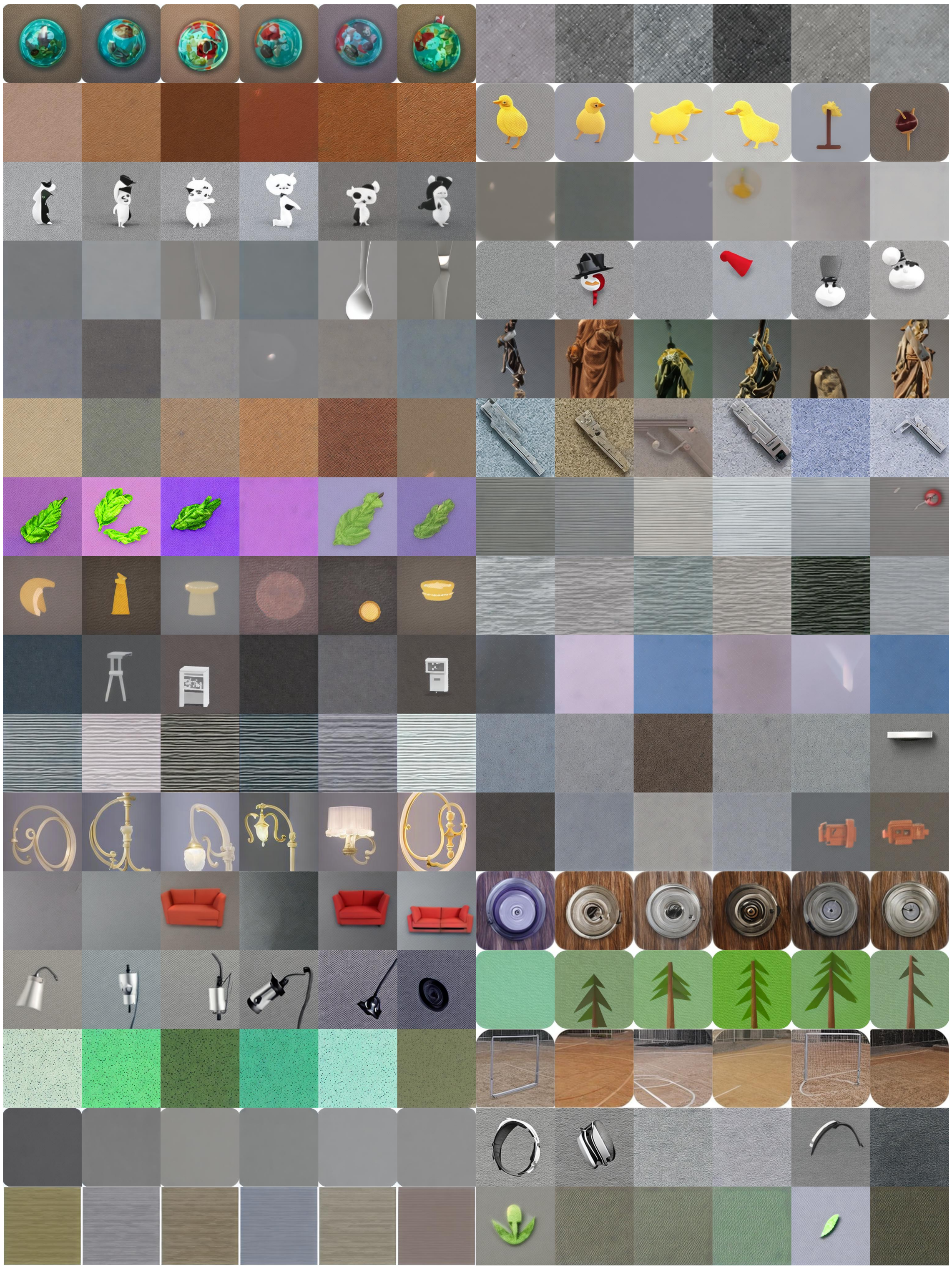


Figure 22: Generated result of the model without training

The model generates images that closely resemble the input image even without training

4.3 Result of the multi-view diffusion model

4.3.1 Qualitative results

The qualitative result of our diffusion model on the validation samples can be found in figure 23. We also include a subsample from Google Scanned Objects [3] which we used in the quantitative analysis below in figure 24

4.3.2 Quantitive results

In addition to qualitative results, we also quantitatively evaluated the performance of our model on the Google Scanned Object dataset [3]. We randomly selected 30 objects and used the rendering code we used to prepare our dataset to render the objects from 12 views. We then evaluate the performance of our model by comparing the generated images with the ground truth images with Zero-1-to-3 [12], SyncDreamer [13] and Wonder3D [14] by measuring the PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index) and LPIPS (Learned Perceptual Image Patch Similarity) scores. The results are shown in table 1.

	PSNR ↑	SSIM ↑	LPIPS ↓
Zero-1-to-3[12]	19.27	0.724	0.180
SyncDreamer[13]	20.10	0.792	0.152
Wonder3D[14]	26.44	0.935	0.052
Ours	25.27	0.827	0.077

Table 1: Quantitative results of the diffusion model on Google Scanned Object dataset

4.3.3 Performance

We applied several techniques to speed up the inference speed of our diffusion model, including quantization to half-precision, using XFormers [9] memory-efficient attention, and applying the Latent Consistency Model Multistep Scheduler proposed in Latent Consistency Models [15]. These improvements allowed us to generate multi-view color and depth images in 3 seconds on a single NVIDIA RTX2080 Ti GPU, which is nearly 10 times faster than Wonder3D [14] inference speed on the same hardware setup.

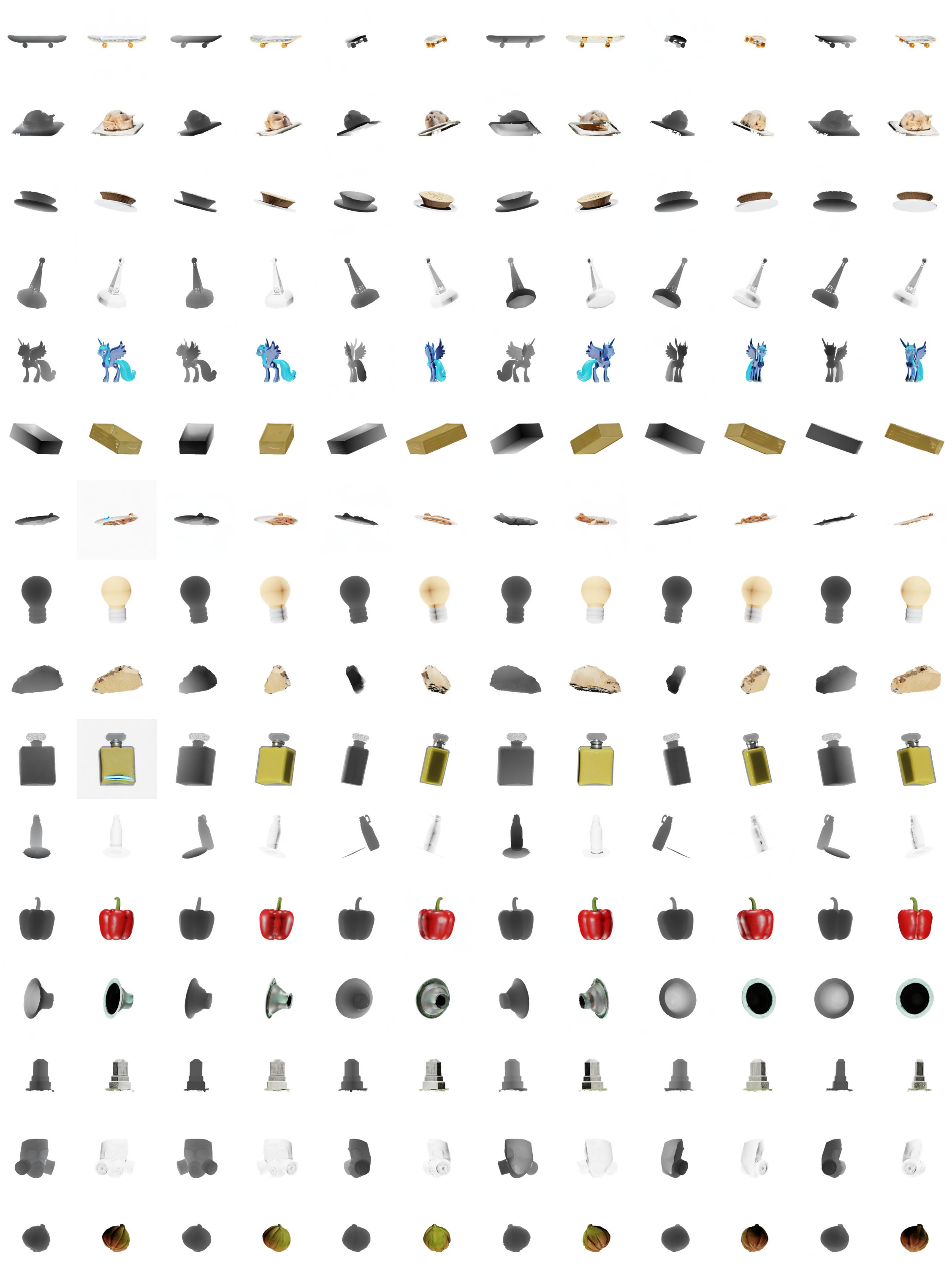


Figure 23: Final result of diffusion model on 16 validation samples

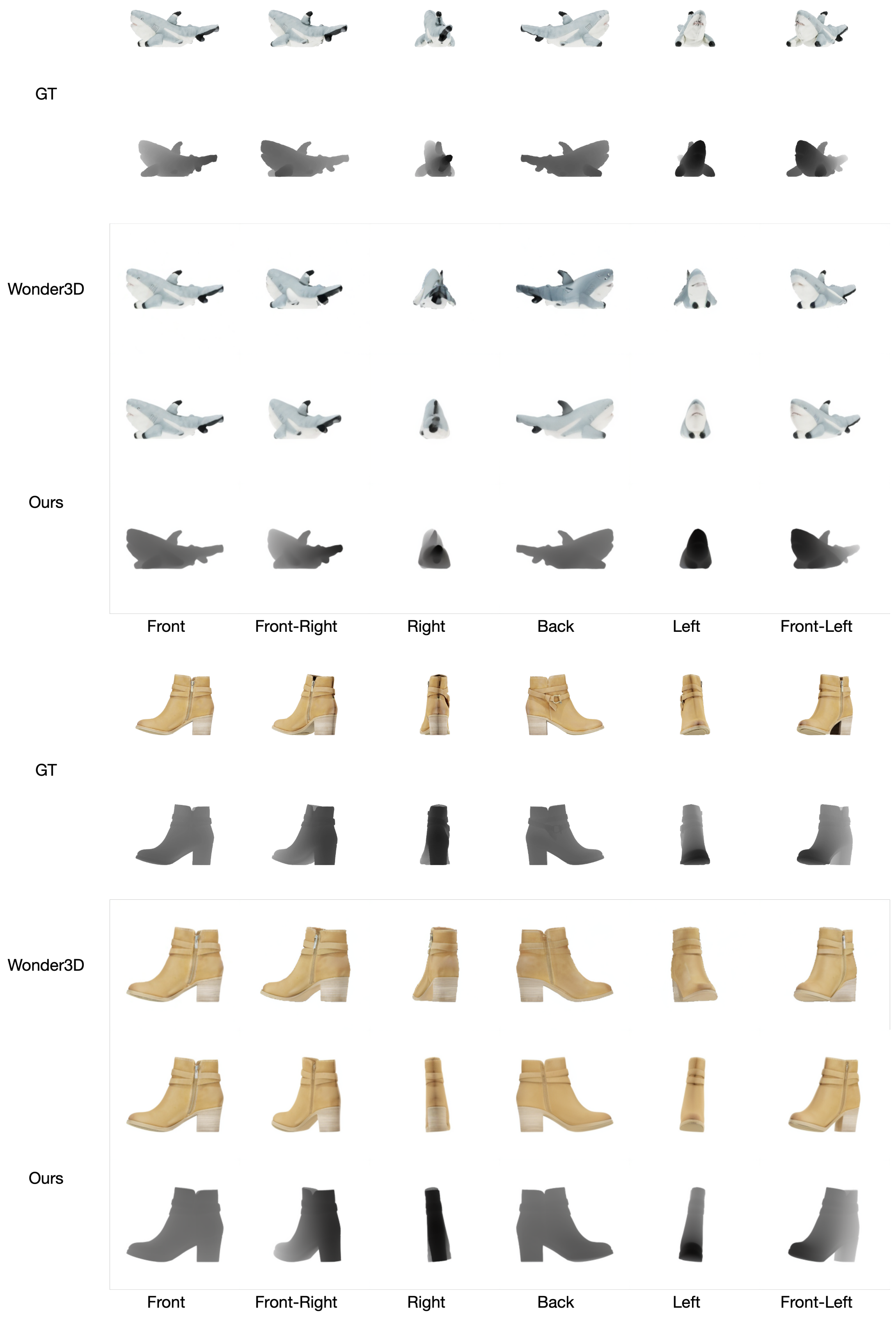


Figure 24: Diffusion model result on subsamples from Google Scanned Objects

4.4 Drawbacks

4.4.1 Sensitive to crop size

We have observed that our model is highly sensitive to the crop size, specifically the ratio of the object occupying the entire image. When the object occupies a large ratio of the entire image, our model tends to produce images like a 2D plane.

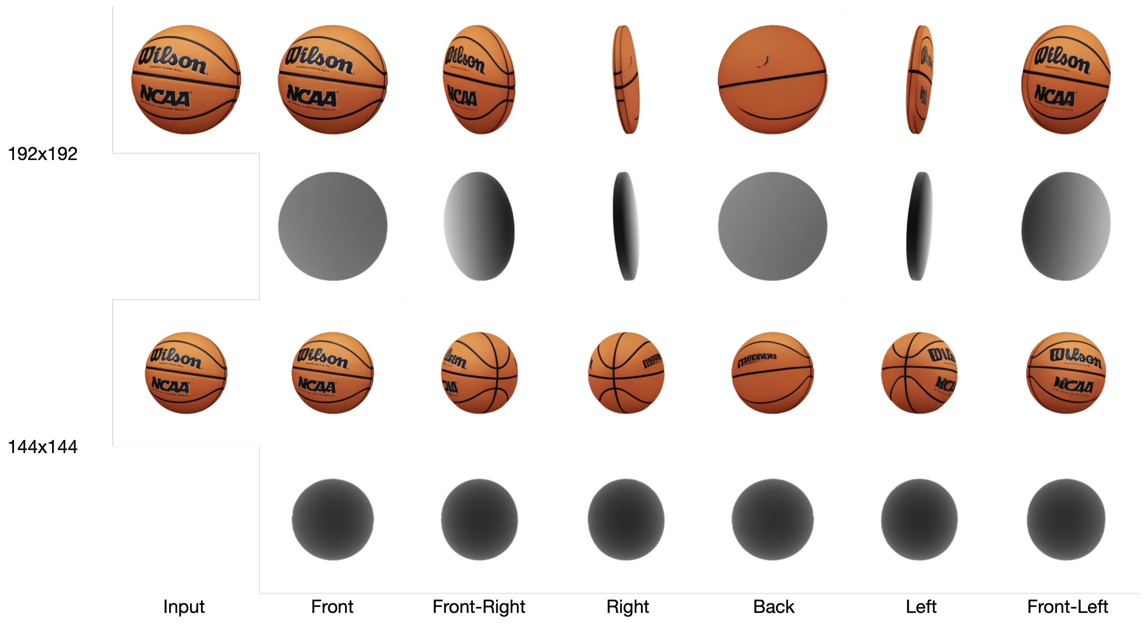


Figure 25: Generated images given different crop sizes

The model tends to produce 2D plane-like images when the object occupies a large ratio of the entire image. Taking this basketball as an example, fixed resolution of 256x256, when the crop size is 192x192 (up), the model tends to treat the object as a 2D plane. When the crop size is 144x144 (down), our model could recover its 3D structure.

After investigation, we found that it might be influenced by a lot of plane-like objects in the training data, as they usually occupy a large ratio of the entire image, and after our random rotation, they look like a thin 2D plane on some sides. To address this issue, maybe more data could be fed into the model, or just clean the data to remove the plane-like objects.

4.4.2 Hard to recover text on the object

We also discovered that due to the nature of the diffusion model, the input image is encoded into a latent space as a conditioning. Therefore, small texts on the object can only be recovered by guessing when minimizing the loss during the training process. We've verified that this problem also occurs in other diffusion models, such as SyncDreamer [13] and Wonder3D [14]. This issue is illustrated in figure 26.

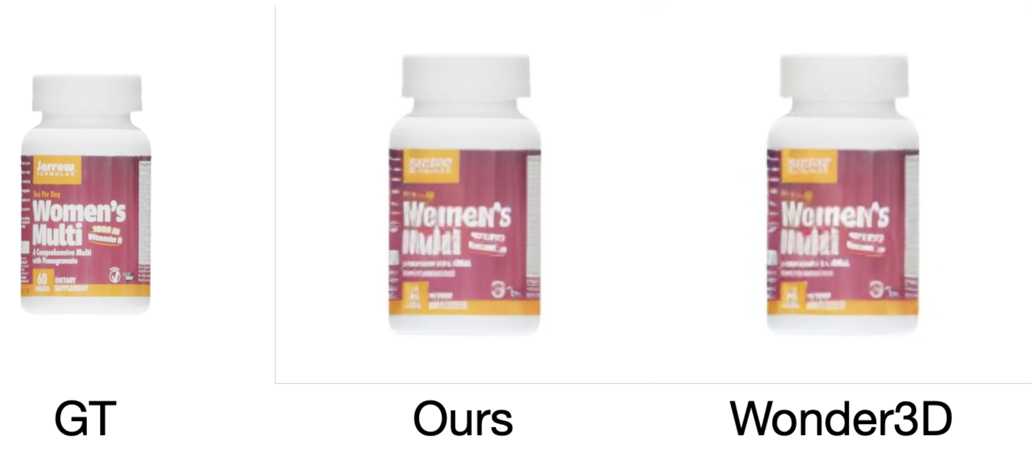


Figure 26: Hard to recover text on the object

The model struggles to recover the text on the object, this is only the front view, and the text is not readable in the generated images

We also investigated this issue on OpenAI's DALL-E [20] by prompting the model "Generate an [DESC] image with text [TEXT] in the bottom". The model sometimes could generate the correct image when the text is in English, but it fails and generates random symbols when the text is in other languages. It might simply be because CLIP [19] cannot encode the information of text in the image, and the model could only guess how the text "looked like" based on the loss.

4.5 Result of the reconstruction model

4.5.1 Qualitative results

The qualitative result of our reconstruction model on randomly collected images can be found in figure 27. We noticed that there are holes in the reconstructed models, although it should be a smooth surface and the depth maps do not infer holes. This might be caused by a large weight assigned to the eikonal loss.

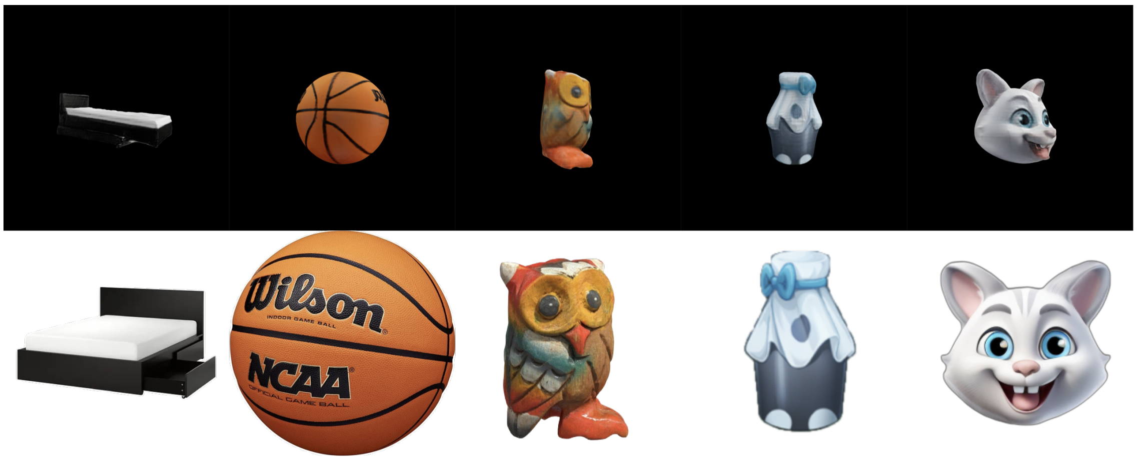


Figure 27: Qualitative results of the reconstruction model on randomly collected images

Meanwhile, we found that our model has a random behavior when doing the reconstruction. The weights of losses may only work for one object but are hard to generalize to other objects, which means that each object may need a different set of weights for the losses, making the reconstruction process more time-consuming and less efficient. In that case, we didn't conduct a detailed qualitative analysis right now.

4.5.2 Performance

By default, for each object, we train our reconstruction model for 5,000 steps, which takes approximately 5 minutes on a single RTX2080 Ti. However, due to the weight of losses issue, we need to tune the weights for each object, which may take a long time to find the optimal weights.

5 Future plan

5.1 Feeding more data to improve the diffusion quality

The current diffusion model is trained on a small dataset, each object selected is rendered only once. We may render the same object with different rotation angles multiple times or just feed in more objects from other datasets. This may also help with the issue of crop size sensitivity we've mentioned in the experiments.

5.2 Diffusion as depth predictor

We found that our model could generate paired depth images given the single color image. In that case, when only considering the front view, we could directly use our model as a depth predictor. More experiments and analyses may be conducted in the future. Also, we may try to abandon the synthesis for multi-view images, while just focusing on depth estimation. We may train a new model using the same architecture while doing depth estimation for the whole scene.

5.3 3D Gaussian Splatting as the 3D reconstruction module

We've already implemented the CUDA kernel of rasterization with depth, as mentioned in the methodology. We may further implement the 3D Gaussian Splatting module and integrate it with the current project.

5.4 Finetuning weights of losses for SDF-based method

We may try to finetune the weights of the losses in the SDF-based method, as mentioned in the experiments. There should be a perfect weight, but it needs more experiments on more objects to find it.

6 Conclusion

In conclusion, this final report outlines the development and result of a framework for 3D reconstruction from sparse image inputs using diffusion models and neural radiance fields. The key objectives are to enable high-quality 3D reconstruction from single images, allow incremental enhancement with additional views, handle real-world inputs flexibly, and analyze tradeoffs compared to state-of-the-art methods.

The literature review summarizes relevant research on diffusion models, neural radiance fields, 3D Gaussian splatting, and recent works combining these approaches for novel view synthesis and 3D reconstruction.

The proposed methodology combines a multi-view diffusion model that generates paired color and depth images and an SDF-based model for 3D reconstruction. We covered detailed training setup in the experiments section, including data preparation, training process, and evaluation metrics. The results show that our diffusion model can achieve competitive performance on the Google Scanned Object dataset. However, the reconstruction module may not be as good as state-of-the-art methods and still needs improvement.

Future work will focus on feeding more data to improve the diffusion model, using diffusion as a depth predictor, implementing 3D Gaussian splatting as the reconstruction module, and finetuning the weights of losses for the SDF-based method.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 6

[2] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. *arXiv preprint arXiv:2212.08051*, 2022. 9, 18

[3] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE, 2022. ii, 24

[4] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5356–5364, 2019. 18

[5] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7441–7451, 2023. 14

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Adv. Neural Inform. Process. Syst.*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. v, 1, 3, 4

[7] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. ii, v, 1, 3, 6, 7, 8, 17

[8] lambdalabs. Stable diffusion image variations. https://huggingface.co/spaces/lambdalabs/stable-diffusion-image-variations. 22

[9] Benjamin Lefaudeux, Francisco Massa, Diana Liskovich, Wenhan Xiong, Vittorio Caggiano, Sean Naren, Min Xu, Jieru Hu, Marta Tintore, Susan Zhang,

Patrick Labatut, Daniel Haziza, Luca Wehrstedt, Jeremy Reizenstein, and Grigory Sizov. xformers: A modular and hackable transformer modelling library. https://github.com/facebookresearch/xformers, 2022. 24

[10] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8456–8465, 2023. 1, 6, 15, 17

[11] Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Mukund Varma T, Zexiang Xu, and Hao Su. One-2-3-45: Any single image to 3d mesh in 45 seconds without per-shape optimization. *Adv. Neural Inform. Process. Syst.*, 36, 2024. v, 11

[12] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9298–9309, 2023. v, 8, 9, 11, 14, 24

[13] Yuan Liu, Lin Cheng, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. SyncDreamer: Generating Multiview-consistent Images from a Single-view Image. *arXiv (Cornell University)*, 9 2023. 9, 14, 24, 28

[14] Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, et al. Wonder3d: Single image to 3d using cross-domain diffusion. *arXiv preprint arXiv:2310.15008*, 2023. v, 10, 11, 13, 14, 15, 16, 24, 28

[15] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023. 24

[16] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. v, 1, 3, 5, 6, 8, 10, 16

[17] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):1–15, 2022. 1, 6, 15, 17

[18] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D Diffusion. *arXiv (Cornell University)*, 9 2022. v, 10

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 5, 28

[20] A. Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. *arXiv (Cornell University)*, 4 2022. 28

[21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10674–10685, 2022. v, 1, 4, 5, 8, 9, 12, 13, 14

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-NET: Convolutional Networks for Biomedical Image Segmentation. *arXiv (Cornell University)*, 5 2015. 4

[23] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv (Cornell University)*, 5 2022. 10

[24] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. Mvdream: Multi-view diffusion for 3d generation. *arXiv preprint arXiv:2308.16512*, 2023. v, 9, 13, 14

[25] Robust Multiview Stereopsis. Accurate, dense, and robust multiview stereopsis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(8), 2010. 1

[26] Engin Tola, Christoph Strecha, and Pascal Fua. Efficient large-scale multi-view stereo for ultra high-resolution image sets. *Machine Vision and Applications*, 23:903–920, 2012. 1

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 6, 14

[28] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NEUS: Learning Neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv (Cornell University)*, 6 2021. 1, 6, 11, 14

[29] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 7