Single-Image 3DGS Scene Reconstruction with Geometry-Aware Priors

Machine Visual Perception Course Project Report

December 4, 2025

Information

Authors: Kacper Michalik, Radhika Iyer, Alex Loh
Group Number: 7
Supervisor: Ahmet Canberk Baykal

Chapter 1: Introduction and Motivation

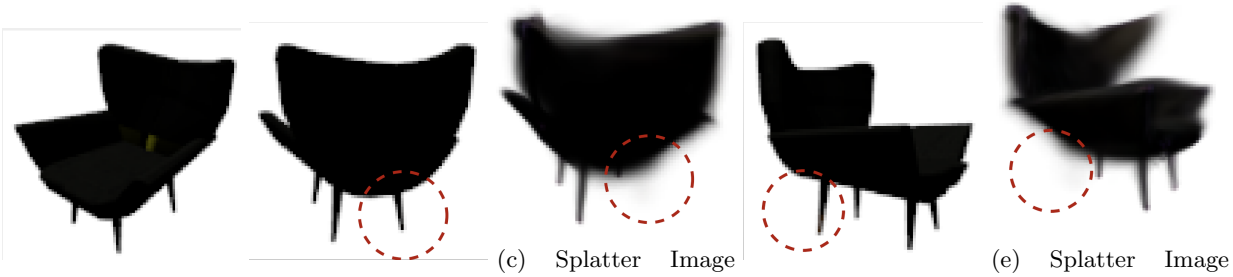
1.1 Introduction to the problem

[Provide a thorough introduction to the problem and why it is important. Briefly explain what general techniques there are and how your project fits.]

The advancement of 3D data acquisition, reconstruction, and rendering methods remains a fundamental and persistent open problem in computer vision. Efficient, high-quality 3D reconstruction is increasingly critical for applications ranging from augmented reality (AR/VR), autonomous devices (for example in perception or navigation in robotics or self-driving cars) to digital artistry (geometry acquisition for models in VFX, games or other digital products), driving significant research interest in this domain.
Historically 3D reconstruction has been a challenging task that requires large number of reference images and even larger amounts of compute. Advancements in computer hardware and machine-learning methods have significantly improved the efficiency and accuracy of 3D reconstruction, extending its applicability for a wider variety of tasks and hardware platforms; continuing this trend, one area of focus is made on further reducing the number of input images required for reconstruction, thereby improving computational efficiency and reducing data acquisition hardware requirements, this has lead to interest in the issue of novel view synthesis. Namely, in 2020 Neural Radiance Fields[1] (NeRF) introduced a revolutionary method for high quality novel view synthesis. This is done by learning a continuous volumetric density and radiance function, which can then be ray-marched from new camera poses, enabling novel views to be synthesized. In 2023, 3D Gaussian Splatting[2] (3DGS) introduced an alternative method, offering major improvements in computational performance. Instead of an implicit function, 3DGS optimizes a set of 3D Gaussians, defined by parameters such as position, colour, opacity, rotation and scale, which can be efficiently rasterized to generate novel views. Developments in 3D Gaussian Splatting methods have allowed for 3D scene reconstruction using few or even single RGB images. While faster than other scene reconstruction techniques and requiring only a "one-shot" pass, these approaches often suffer from challenges such as layout/scale drift, over-smooth geometry and hallucinations in occluded regions [3] (NOTE TODO: for now this only talks mostly about over-smooth geometry and kind of hallucinations, still looking for a layout/scale drift one).

This project focuses on one recent method, Splatter Image[4], as a baseline. Splatter Image allows single or few RGB image 3DGS reconstruction. Achieved by predicting 3D Gaussians as pixels in a multichannel image; this representation reduces reconstruction to learning an image-to-image neural network, allowing the use of a 2D U-Net to form the representation. Each pixel stores the parameters for a corresponding 3D Gaussian, allowing for reconstruction in a single feed-forward pass. This overall architecture allows

for a compute-efficient model. Despite its speed, Splatter Image does have some issues that have been noted in related works, particularly in reconstructing structures unseen in the input view, including for views significantly different from the source. One source of this problem may be that the prediction of 3D Gaussians in Splatter Image is based on input image features alone, which do not have sufficient conditional information for Gaussian splatting to represent structures that are not visible in the input view[5]; shown in figure 1, Splatter Image has trouble generating the occluded chair leg in 1c and 1e.



(a) Input image (b) Ground truth (c) Splatter Image output (d) Ground truth (e) Splatter Image output

Figure 1: Splatter Image outputs compared with ground truth taken from [5]

This project aims to address this by first researching inferable geometry priors (such as planes, normals, visibility cues, depth, segmentation or edge maps) which can be dynamically produced for input images by existing specialized models, then proposing a lightweight augmentation for Splatter Image, allowing predicted priors to be fed alongside the RGB images, allowing them to guide reconstruction in a more accurate manner compared to current case, where the neural network needs to learn these geometric features itself.

1.2 Background and related work

[Include a few very relevant related works and how your work relates to those, expanding on the previous section. We do not expect you to cover all previous works.]

ADD DIAGRAMS OF PHTOGRAMETRRYT GAUSSAIN SPLAT AND NERF

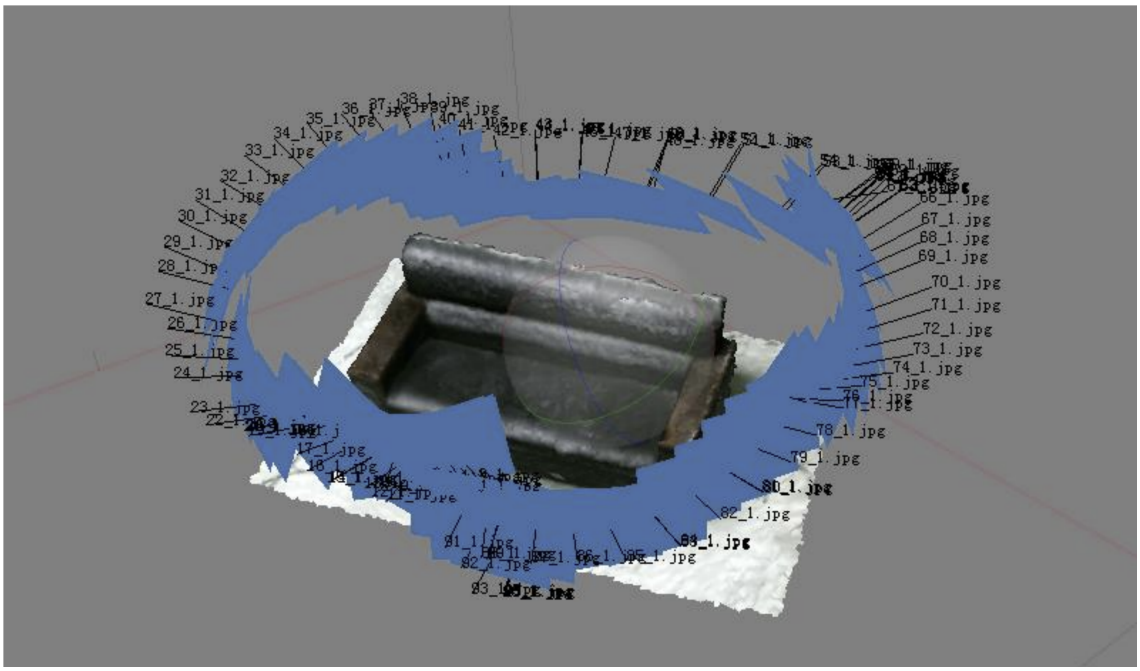


Figure 2: An example of 3D reconstruction with many overlapping images from [6]

Traditionally, 3D reconstruction has been performed by multi-stage photogrammetry pipelines, relying on explicit geometric representations such as meshes or point clouds. The industry-standard workflow begins with Structure from Motion (SfM), which matches sparse feature points across many overlapping images to estimate camera parameters and generate a sparse point cloud. This is typically followed by Multi-View Stereo (MVS) algorithms to compute dense depth maps, which are combined to generate a standard 3D mesh using techniques like Delaunay triangulation[7] or Moving Least Squares with Marching Cubes[8]. While effective for static, diffuse environments, these methods struggle significantly with surfaces such as transparent windows or reflective metals, this as they rely on strict photometric consistency and lack a mechanism to deal with view-dependent radiance. Additionally, these methods require large numbers of high-resolution images with substantial overlap to achieve high-quality reconstruction. This heavy data acquisition requirement means the methods create a computational bottleneck, requiring hours of processing time on high-end hardware, and are impractical in the first place without complex image acquisition setups, such as in Figure 2.

A paradigm shift occurred in 2020 with the introduction of Neural Radiance Fields[1]. NeRF moves away from explicit geometry representations to an implicit volumetric representation. In NeRF an underlying continuous volumetric scene function, represented using a deep learning model, is optimized using a set of input images with known camera poses; the input to the function/model is a single continuous 5D coordinate (spatial location and viewing direction) and the output is the volume density and view-dependent emitted radiance at that location. Novel views are synthesized by querying 5D coordinates along camera rays for some new camera pose (ray-marching), and output colours and densities are rendered into an image. NeRF allowed for higher quality 3D reconstruction, eventually using sparser image sets (PixelNeRF, RegNeRF), compared to traditional photogrammetry methods, achieving state-of-the-art results and becoming the gold standard for novel view synthesis. TODO: CITE

In 2023 the field was further revolutionized by 3D Gaussian Splatting, offering a computationally high-performance alternative to NeRF. 3DGS offers a new explicit geometry representation, modelling a scene as a collection of parameterized 3D Gaussians[2]. Unlike NeRF, 3D Gaussian Splatting does not rely on a neural network to generate a scene. Instead, in the original paper, reconstruction is achieved by first initializing a set of Gaussians, either randomly or from a sparse point cloud (typically from Structure from Motion), where each Gaussian is defined by a set of parameters: position (mean), covariance (scale and rotation), opacity, and colour (represented using Spherical Harmonics to allow for view-dependent radiance). Then an optimization process is run that first adjusts the Gaussians' parameters to minimize the error between ren-

dered images and the ground truth; and secondly performs dynamic management of the density of Gaussians within the scene, by splitting, cloning or pruning Gaussians in an interleaved manner; namely Gaussians in under-reconstructed areas are cloned, Gaussians with high variance are split, and Gaussians in areas of low opacity and of excessive size are pruned. Finally, for rendering, the Gaussians are rasterized using a custom tile-based rasterizer to produce resulting views, allowing millions of Gaussians rendered in real-time, allowing for real-time novel view synthesis. While the original method relies on per-scene optimization to generate 3D Gaussian Splats, recent works have begun using deep learning methods to predict sets of Gaussians directly, with most recent developments allowing for 3D scene reconstruction using few or even single RGB images.

ADD IAMGES OF F3D GAUS, GAUSS VIDEODREAMER, TGS, SPLATTERIMAGE

Since the introduction of 3DGS, a number of deep learning architectures and processing pipelines based on the method have been developed to find the most accurate and efficient implementation capable of producing high quality 3D Gaussian Splats. Recent examples include ExScene, Wonderland, F3D-Gaus[9], Gauss VideoDreamer[10], TGS and Splatter Image[4].
F3D-Gaus
Gauss VideoDreamer
The triplane representation was proposed to efficiently and expressively represent 3D volumes [11], as a compromise between rendering speed and memory consumption. They were shown to scale to large datasets like Objaverse [12][13], but at the cost of hundreds of GPUs for multiple days [14].
Another recent method is Splatter Image [4] then applies Gaussian Splatting to monocular reconstrution by using a set of 3D Gaussians as the 3D representation. It predicts a 3D Gaussian for each of the input image pixels and uses a 2D image as the container of the 3D Gaussians, storing the parameters of one Gaussian per pixel. This reduces the reconstruction problem to learning an image-to-image neural network, allowing the reconstructor to be implemented utilizing only efficient 2D operators. The use of Gaussian Splatting in this approach increases rendering and space efficiency, which benefits inference and training. Our work continues to expand on this method through investigating different geometry priors and integrating them into the current model as appropriate.

1.3 Overview of the idea

[Provide an overview stating why the idea of the project makes sense and what the main motivation is.]

whilst providing excellent quality recsontruction from popint of vie wof refernec iamge, the actual 3d reocusntrion and resulting novel views suffer from hallucinations in the form of floating geometry, etc etc, this is fundamanetally due to the lack of inforamtion the model has from a single input image feeding models additional data improves reconstruction with modern compute and ml advancements there now exist many good quality pretrained geometry related models for example generating depth, normal maps, segmentation we propose exploting the knowledge/capacibiltiy of these models to predict additional priors of input images, then creating a modified model that accepts these priors, allowing priors to guide reconsutrcution, improvingt recosntructio quality we also propose performing ablation study to see which priors are most effective/significant in changing the reconstruction quality

Chapter 2: Method

2.1 Baseline algorithm

[Explain the baseline architecture you used to build your algorithm on. You may reproduce figures from the original papers.]

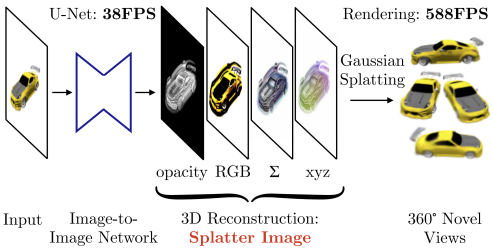


Figure 3: Overview of SplatterImage[4]

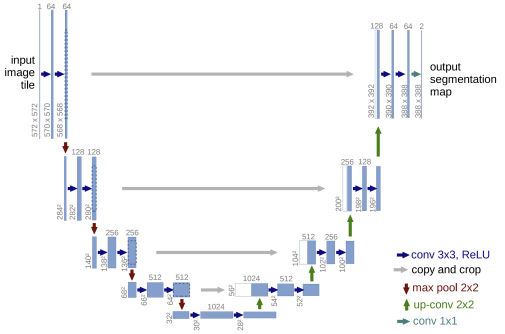


Figure 4: U-net architecture[15] that Song U-Net[16] is based on

Splatter Image uses a standard image-to-image neural network architecture to predict a Gaussian for each pixel of the input image I, generating the output image M as the Splatter Image. Learning to predict the Splatter Image can be done on a single GPU using at most 20GB of memory at training time for most single-view reconstruction experiments (except for Objaverse, where 2 GPUs were used and 26GB of memory was used on each). Most of this neural network architecture is identical to the SongUNet of [16], but the last layer is replaced with a 1×1 convolutional layer with $12 + k_c$ output channels, where $k_c \in \{3, 12\}$ depending on the colour model. The output tensor codes for parameters that are then transformed to opacity, offset, depth, scale, rotation and colour respectively. These parameters are then activated by non-linear functions to obtain the Gaussian paramters, such as the opacity and depth. The Gaussian Splatting implementation of [2] is used for rasterization to generate 360° views of the original input image.

TALKA BOUT ARCH MORE, CROSS ATTN FILM, Each blue box corresponds to a multi-channel feature map, with the number of channels denoted on top of each box. The x-y size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

2.2 Algorithm improvements

[Explain what you implemented to improve over the baseline. You may include figures to explain the idea and logic. Focus on the ideas and not the implementation.]

explain insertion of additional layers explain addition of transofrmer/FiLM layers to allow multimodal input from segmentation tokens

FiLM layers learn by training a separate, small neural network called a FiLM generator. This is a new, small network (like an MLP or RNN), segmentation embedding is then fed into the FiLM generator The generator outputs a gamma (scale) and beta (shift) The gamma and beta values are applied to intercepted alyer using the FiLM equation: y = gamma * x + beta This modulated feature map is passed to the next layer of the U-Net. Exisitng U-Blocks support Film for camera embeddings, we modidy U-net blocks to have additional film layer for segemtnation tokens

Add new cross-attention modules inside the U-Net's blocks (in the bottleneck and decoder). The U-Net's image features act as the Query (Q). The multimodal input embeddings act as the Key (K) and Value (V). More expressive than FiLM as it allows for fine-grained spatial conditioning (per-pixel)., this method is consdiered state-of-the-art for high-performance conditional generation.

explain addition of LORA matrices due to compute limitations/allowign working of exisitng model

2.2.1 Low-Rank Adaptation (LoRA)

To address the computational constraints of fine-tuning the full U-Net architecture, we integrated Low-Rank Adaptation (LoRA) [17] directly into our GaussianPredictor. While our implementation shares a similar class structure to the official `loralib` library [18] (utilizing mixins to wrap `Linear` and `Conv2d` layers), we manually adapted the forward pass to support the specific channel dimensions of the Splatter Image architecture (further detail is in **Section X: Manual LoRA Integration**).

Adapters are small modules placed after the frozen modules we wish to adapt, such as linear and convolutional layers. The adapter accepts the same input dimension as the original layer and produces the same output dimension. This allows the output from the adapter to be summed element-wise with the frozen layer's output.
Instead of learning a new large weight matrix for these adapters, the weight update is decomposed into two smaller low-rank matrices. For a weight matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$, the update is defined as $W + \Delta W$, where ΔW is factored into:

$$A \in \mathbb{R}^{r \times d_{in}} \quad \text{and} \quad B \in \mathbb{R}^{d_{out} \times r}$$

Here, we see that $r \ll \min(d_{in}, d_{out})$, where r is the rank hyperparameter (detailed below). During training, only the parameters of A and B are updated, while the original weights W remain frozen.
For `Conv2d` layers, we treat the kernel $W \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$ as a flattened matrix of shape $C_{out} \times (C_{in} \cdot k \cdot k)$. The flattened representation is decomposed into B and A, allowing us to apply LoRA to the full spatial kernel. By including the spatial dimensions $(k \times k)$ in the decomposition, the adapter can learn spatial feature refinements, rather than being limited to channel-wise linear projections. This approach follows the implementation found in `loralib` [18].

During the forward pass, the input x is processed by both the frozen weights W and the LoRA branch:

$$h = Wx + \frac{\alpha}{r} BAx \tag{1}$$

where $\frac{\alpha}{r}$ is a scaling factor [18]. This scaling normalises the updates across different rank choices, reducing the need to re-tune the learning rate when r changes.

We utilise three hyperparameters to control this adaptation:

- Rank (r): The rank of the low-rank matrices A and B. Higher ranks increase the number of parameters and the capacity of the adaptation, but also increase computational cost
- Alpha (α): A scaling factor applied to the LoRA update during the forward pass. The update is scaled relative to the weights that have been frozen from the pretrained model.
- Dropout (p): The dropout probability applied to the LoRA layers during training. This randomly disables activations, which aims to prevent overfitting.

Following [18], A uses Kaiming uniform intialisation, and B is initialised to zero. This ensures that $\Delta W = 0$ at the start of training, which preserves the behavior of the pre-trained model initially.

2.3 Implementation details

[Explain how you implemented the improvements. You may include code snippets with the corresponding explanations.]

All code associated with the project can be found in the following repositories:

3DGS-priors (Top level repository for the project): `https://github.com/Kacper-M-Michalik/3dgs-priors`
Splatter Image Fork: `https://github.com/Kacper-M-Michalik/splatter-image`

Generated datasets and model weights can be found in the following repositories:

Datasets with Predicted Priors: `https://huggingface.co/datasets/MVP-Group-Project/srn_cars_priors`
Pretrained Models: `https://huggingface.co/MVP-Group-Project/splatter-image-priors`

2.3.1 Planes and Normal Maps Exploration

WE CONSDIERED PROVIDING MODEL WITH SPTAIL INFORAMTION ABOTU THE SCENE AS BEING ONE OF THGE MSOT LIEKLY AENUES OF IMPROVEMENT, THIS CAME IN 2 FLAVOURs, PREDICITNG PLANES AND STREUCTURES IN SCENE AND SURFACE NORMAL MAPS

WHEN CONSDIERING GENERTING PLANES For example teddybears (as seen on the CO3D Teddybears) do not have dominant planes, they are complex convex shapes, using a planar prior might confuse the network or cause it to flatten the bear's features; as such we decided against using planes as a prior.

A much more favourable option were normal maps. Normal maps store surface normal data as RGB colour information,

showing how light interacts with the surface at a per-pixel level, hence we wanted to investigate if that could help the model generate the 3D surface of the input image.

For our ground truths we use [19] which provides a dataset of higher resolution images of the ShapeNet models from [20] each paired with a depth image, a normal map and an albedo image at `https://github.com/Xharlie/ShapenetRender_more_variation`. We then feed these images into the normal map generation models and compare against the ground truth normal maps to evaluate their performance.

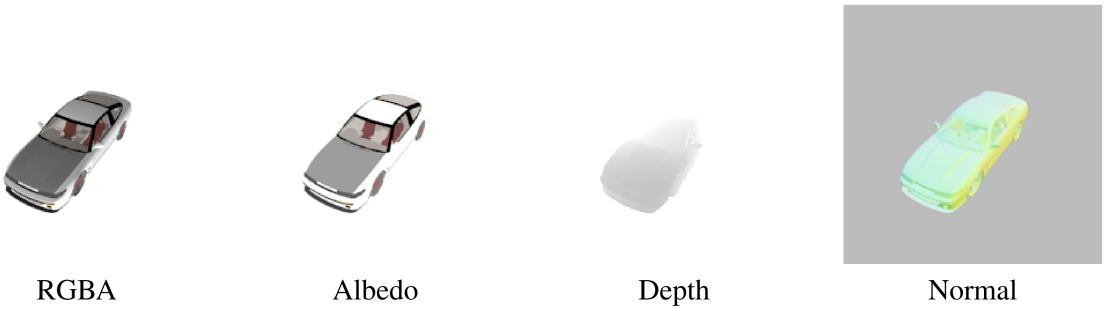
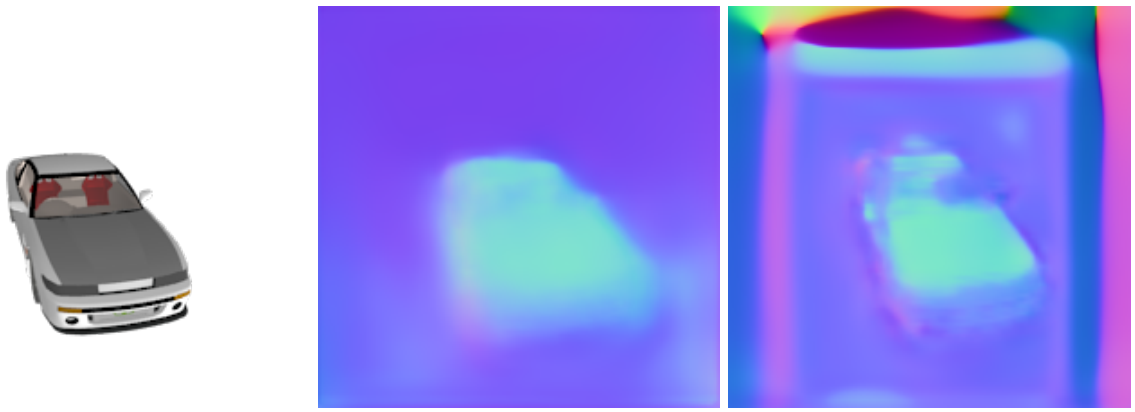


Figure 5: Example of image with maps used as ground truth taken from [21]

For the models we referenced [21] which implements a network which estimates the per-pixel surface normal probability distribution and uses uncertainty-guided sampling to improve the quality of prediction of surface normals. The paper provided code at `https://github.com/baegwangbin/surface_normal_uncertainty` that implemented this method on a network trained on ScanNet [22], with the ground truth and data split provided by FrameNet [23], and another trained on NYUv2 [24], with the ground truth and data split provided by GeoNet [25] [26]. Both models take in the original image and dimensions of the image as input and return a corresponding normal map with the same dimensions as the given input dimensions.

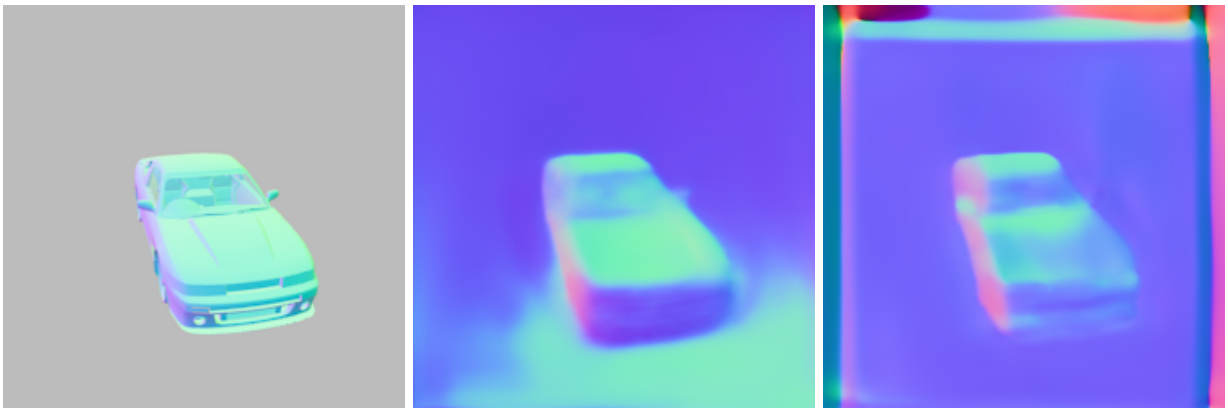
We run both pretrained models on the dataset.



(a) Original (b) ScanNet output (224x224) (c) NYUv2 output (224x224)

Figure 6: Comparison of original input and two model outputs

We then pass in input dimensions larger than the actual ones into the models, such that a normal map larger than the original input is produced. We then resize the image to the original input dimensions.

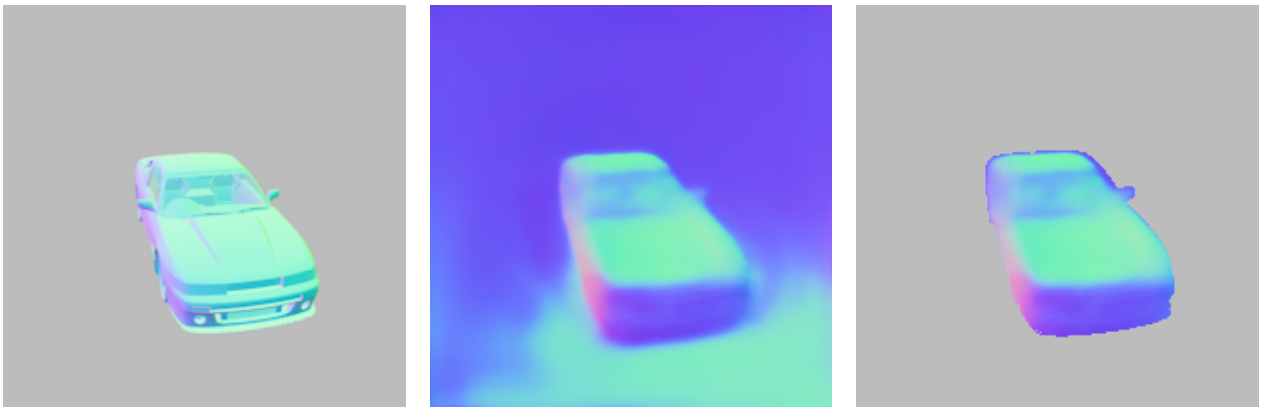


(a) Ground truth from [19] (b) ScanNet output (448x448) (c) NYUv2 output (448x448)

Figure 7: Comparison of model outputs when setting input dimensions as 448x448 instead of 224x224 alongside ground truth

The normal map generated for images when given larger input dimensions seem to have more clearly defined edges and surface contouring. It is also important to note that the ground truth for NYUv2 is only defined for the centre crop of the image and the prediction is therefore not accurate outside the centre. This is shown in figures 6c and 7c where noise is generated around the borders of the normal maps.

To compare our generated normal maps to the ground truth normal maps provided in [19], we first mask out the background of the generated normal maps such that the difference in background colour does not contribute to the evaluation metrics for normal map generation.



(a) Ground truth from [19] (b) ScanNet output (448x448) (c) Output with background masking

Figure 8: Example of masking out background for model evaluation against ground truth

We then use Pixel Based Visual Information Fidelity to compare the normal maps generated by the two models to the ground truth. Visual Information Fidelity is a reference image quality metric that quantifies the amount of visual information preserved after image processing [27] and can be used to measure various image quality attributes such as noise level and sharpness [28].

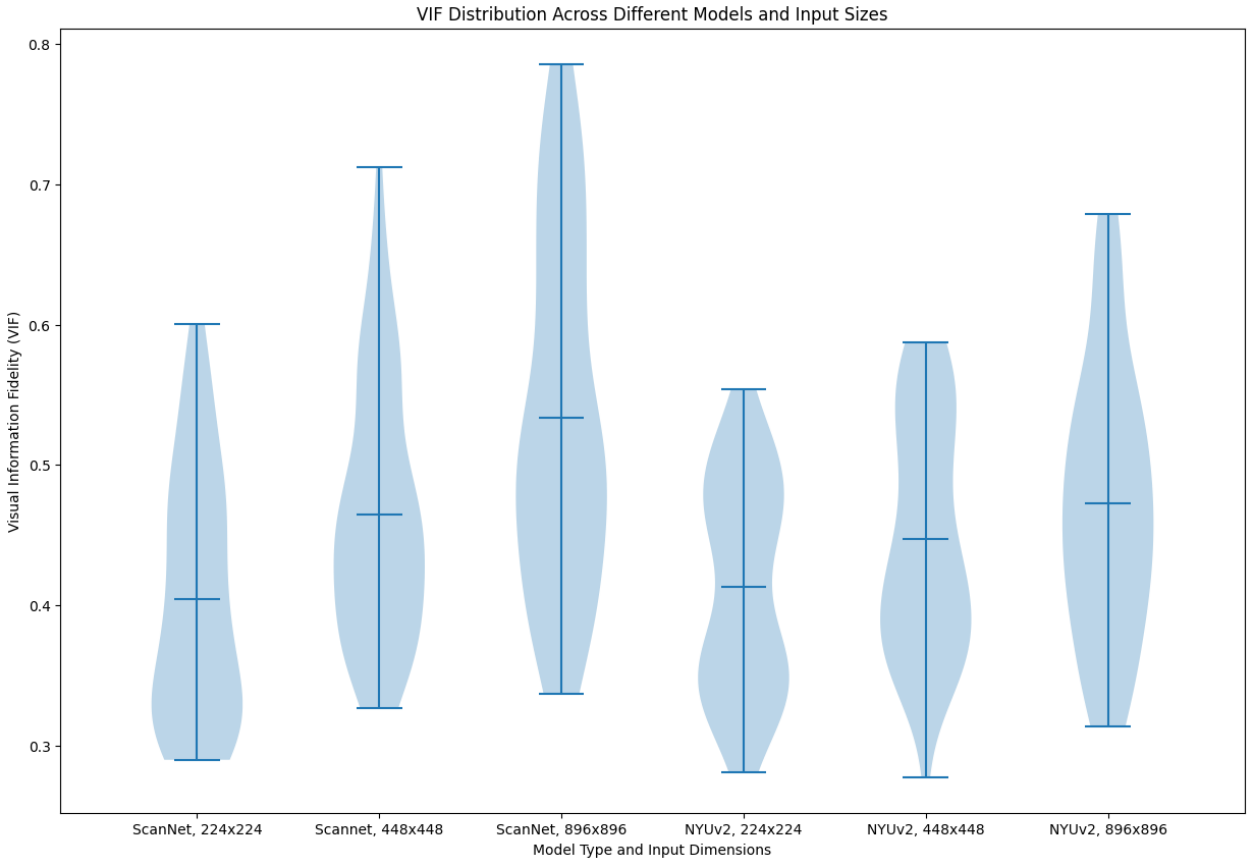
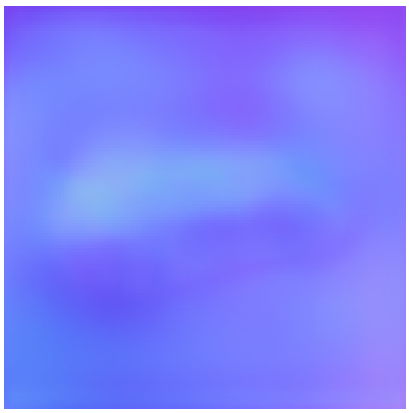


Figure 9: Comparison of VIF between ground truth and different models

From Figure 9 we see that the model trained on ScanNet generates normal maps that are closer to the ground truth compared to that trained on NYUv2 on average. Hence, in the final model we decided to use the model trained on ScanNet on the ShapeNet database in [20].

(a) Original image from [20] of size 128x128



(b) Output from ScanNet model with 128x128 passed in as input dimensions

(c) Output from ScanNet model with 512x512 passed in as input dimensions

Figure 10: Original ShapeNet image and normal map outputs

Without passing in dimensions larger than the input image into the model, we can see from comparing

Figures 6a and 6b to Figures 10a and 10b that the quality of the normal map generated decreases as the resolution of the original input image decreases. Hence, we pass in much larger input dimensions (512x512) to generate a normal map of higher quality, as shown in Figure 10c.

2.3.2 Depth Map Exploration

Depth maps store the distance of a surface from the camera per-pixel. These distances vary in type, such as metric, which considers the physical distance from the camera to the observed point, and relative (such as those produced by the models below). Monocular depth estimation (MDE) models input just a singular image, and produce a depth map (relative distance).

Produced depth maps were compared against the "ground truths" produced by `https://github.com/Xharlie/ShapenetRender_more_variation`, as was done in the normal priors exploration. An example of the depth map produced by them is visible in Figure 5. However, it is important to note that these depth map "ground truths" were not always perfect, as can be seen in the following example:

- (INSERT) Image needed here of poor ground truth.

This inclined us to take the quantitative results produced by comparing MDE models tested against these ground truths with a pinch of salt. For each produced depth map, the following metrics were used to compare against the ground truths.

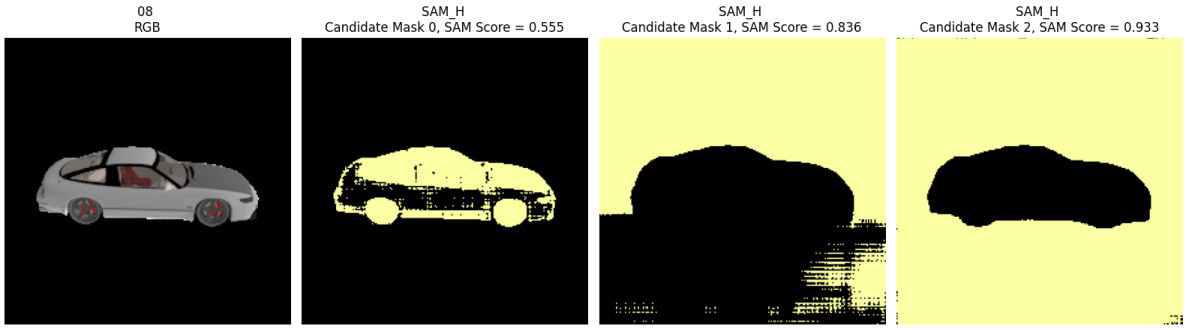
1. **Absolute Relative Error**: Measures the average difference between the predicted depth and the ground truth, normalised by the ground truth depth.
2. **Root Mean Squared Error (RMSE)**: Calculates the standard deviation of the residual errors.
3. **Scale-invariant RMSE (SI-RMSE)**: Computes the RMSE while ignoring the unknown absolute scale and shift between the prediction and ground truth.
4. δ at **1.25** ($\delta_{1.25}$): Represents the percentage of predicted pixels p that satisfy the condition $\max(\frac{p}{p^{gt}}, \frac{p^{gt}}{p}) < 1.25$, which takes into account close pixel-wise agreement.

The following table (LINK to label) summarises the mean metrics across the MiDaS models tested.

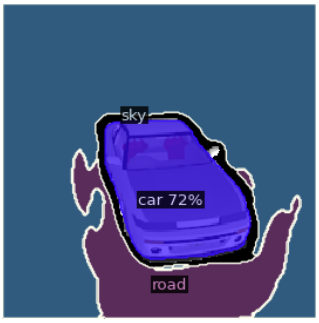
2.3.3 Segmentation and Salient Object Detection Exploration

Separating pixels belonging to the foreground object, through a segmentation mask or, as will be detailed below, using a salient object detection (SOD) model, can be another prior. This involves producing a binary mask that separates an object from its background.

Initially, we explored standard semantic and panoptic segmentation models, such as those found in the Detectron2 [29] model zoo, and the Segment Anything Model (SAM) [30]. These models are often used for segmentation, but as illustrated in Figure 11, these produced non-contiguous masks that often had sections that included more background pixels. Segmentation models are also limited on their training classes, and despite being tested on categories in this set, their masks were improved on by salient object detection models.



(a) SAM Results (3 candidate masks produced per input)



(b) Panoptic Segmentation (Detectron2 Model) Result

Figure 11: Sample outputs from standard segmentation approaches.

SOD models identify the most visually distinct object in a scene, which allows producing a binary mask that tightly hugs the object boundary.

To quantitatively evaluate SOD models, we noted that the ShapeNet images used (the same as in the normal and depth priors section) had transparent backgrounds, allowing using the alpha channel to be used as the 'ground truth' for the object silhouette.

We tested three SOD architectures: `rembg` (based on the U-2-Net architecture) [31], `InSPyReNet` [32], and `BiRefNet` [33]. We evaluated performance using Mean Absolute Error (MAE), Intersection over Union (IoU), and F_β-Measure.

F_β is a weighted harmonic mean of precision and recall, defined as:

$$F_\beta = \frac{(1+\beta^2)\,\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \tag{2}$$

It is commonly used is salient object detection to assess the quality of binary masks produced. When $\beta = 1$, precision and recall are equally weighted. Greater values of β prioritise recall, while lower ones prioritise precision. We set β^2 to 0.3, following conventional practice in SOD literature, to emphasise precision over recall [34], [35], [36]. When identifying a single salient object, false positives (background pixels incorrectly classified as foreground) are often considered worse than small false negative sections along the boundary of the object.

Note that the ShapeNet images used are split into 'Easy' and 'Hard' as categories, as before.

Table 1: Comparison of Salient Object Detection models on ShapeNet renders. ↑ indicates higher is better, ↓ indicates lower is better.

Difficulty	**Model**	**IoU ↑**	F_β ↑	**MAE ↓**
Easy	rembg (U-2-Net)	**0.986**	0.991	**0.004**
	InSPyReNet	0.983	**0.996**	**0.004**
	BiRefNet	0.966	0.979	0.006
Hard	rembg (U-2-Net)	**0.980**	0.988	**0.005**
	InSPyReNet	0.966	**0.991**	0.006
	BiRefNet	0.952	0.973	0.007

Figure 12 illustrates the distribution of F_β scores across both "Easy" and "Hard" datasets. `InSPyReNet` has the tightest interquartile range, particularly on the Easy set. `rembg` demonstrates similar stability but with a slightly broader spread on the 'Hard ' dataset. `BiRefNet` is competitive (note the scale of the y-axis, with all achieving scores greater than 0.95), but comparatively shows a lower median score and higher variance, suggesting it is more sensitive to specific geometries.

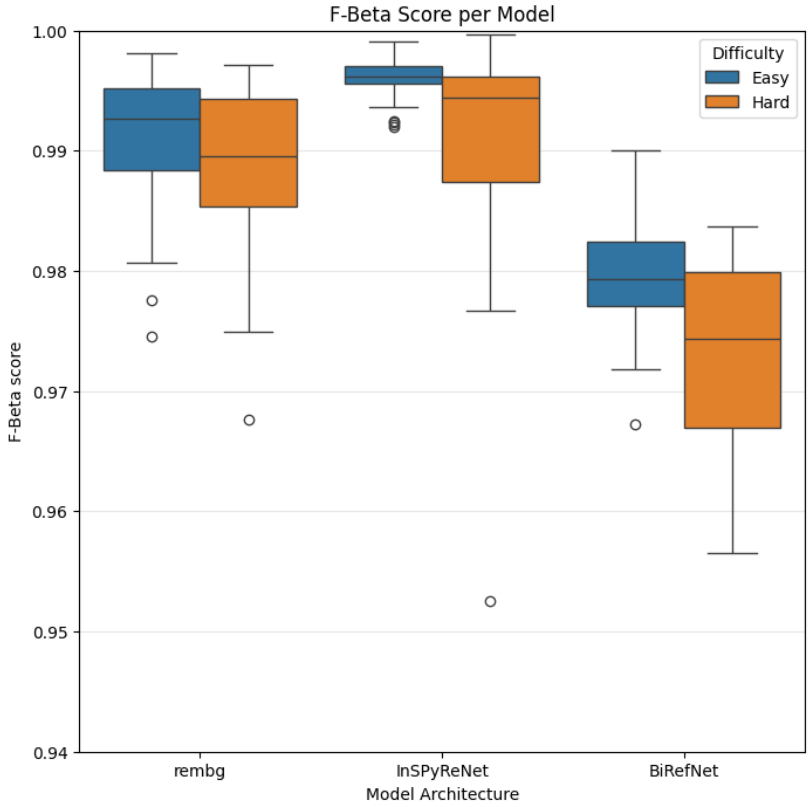


Figure 12: Distribution of F_β scores for each model across the two difficulty levels.

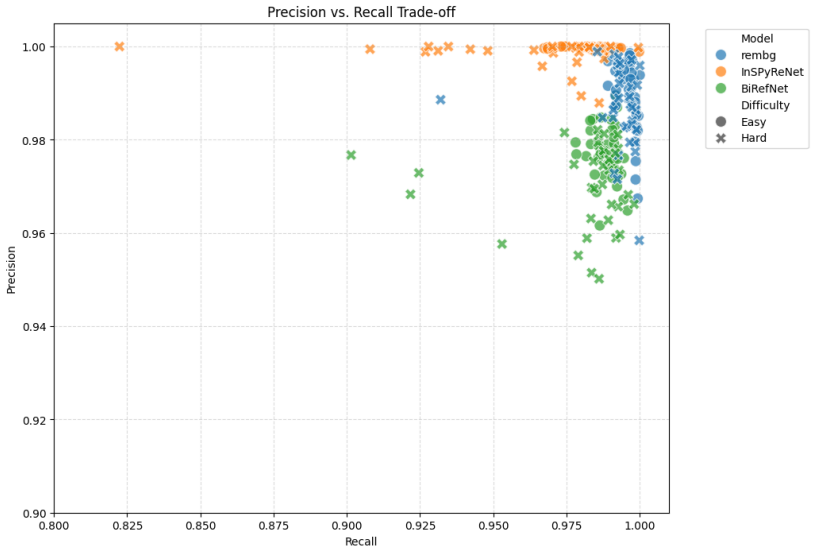


Figure 13: Precision vs. Recall trade-off

The scatter plot in Figure 13 shows how the models operate with different aims; . `InSPyReNet` (in orange) clusters tightly towards the top of the high-performance region (top right), suggesting a priority of precision (often very close to 1.0). In practice, this may mean eliminating some background noise, but this may mean missing parts of the object. `rembg` (blue) also has high recall and precision, and it could also be used as a robust general-purpose model for segmentation (that does not clip parts of the object of interest).

To identify the most frequent "winner," we counted the number of images where each model achieved the highest F_β score. `InSPyReNet` achieved the highest score on the majority (28 vs `rembg`'s 8).

However, to contextualise these results, all three models achieved very high scores (with $F_\beta > 0.97$ and IOU > 0.95), largely as the ShapeNet images have a clear foreground object against a uniform background. This reduces the number of possible cases of background vs foreground confusion, which means the models differ meaningfully here on fine structures at the object boundary. In practice, these can include wing mirrors, antennae, etc. Therefore, the differences in the models come from how well they capture these fine details.

2.3.4 Selected Prior Integration

During model selection, basic notebooks were written to operate and evaluate the relevant models. Once a model was selected, the corresponding notebook would be refactored into a high-performance script designed to process full datasets, producing ready-processed priors for every RGB image within the given dataset. Specifically, inference code would be rewritten to ensure it operates on high performance hardware such as the GPU, and that all operations were executed in a batched manner.

```
with torch.no_grad():
    for batch_imgs, batch_filenames in loader:
        # Prediction, B C H W format
        batch_imgs = batch_imgs.to(device)
        preds = model(batch_imgs)

        # Batched downsize
        preds = torch.nn.functional.interpolate(
            preds.unsqueeze(1),
            size=target_size,
            mode="bicubic",
            align_corners=False
        ).squeeze(1)

        # Batched 0 to 1 normalization
        batch_flat = preds.flatten(start_dim=1)
```

```
        min_val, max_val = torch.aminmax(batch_flat, dim=1, keepdim=True)
        min_val = min_val.view(preds.size(0), 1, 1)
        max_val = max_val.view(preds.size(0), 1, 1)
        preds_normalized = (preds - min_val) / (max_val - min_val + 1e-8)
        preds_uint8 = preds_normalized.mul(255).byte().cpu().numpy()

        # H W format -> 128 * 128
        for j, file_id in enumerate(batch_filenames):
            full_batch.append(ProcessedImage(
                uuid=uuid,
                file_id=file_id.item(),
                image=preds_uint8[j].tobytes()
            ))
```

Listing 1: Batched processing of depths

An example of a performant rewrite is that of the depth generation python script. As shown in the excerpt, inference operates on batches supplied by a DataLoader, with all other operations also being executed batch-wise on the device (the GPU in our case); only once the batch is fully processed is it moved back to regular (host) memory, and the individual priors extracted with relevant metadata for use/saving. We also note that predicted depths are quantized to 8-bit unsigned integers, all predicted priors are quantized this way; for example normals are quantized from 3 * 32-bit floats to 3 * 8-bit unsigned integers; this is due to compute and storage limitations. For example the SRN cars dataset contain 387,956 images, each 128 * 128 pixels, storing only priors such as depths and normals as 32-bit floats for each image would require:

$$\begin{aligned} \textbf{Depths:} \quad & 387,956 \times 128 \times 128 \times 4 \text{ bytes} \\ & = 25,425,084,416 \text{ bytes} \approx \textbf{25.43} \text{ GB} \end{aligned}$$

$$\begin{aligned} \textbf{Normals:} \quad & 387,956 \times 128 \times 128 \times 3 \times 4 \text{ bytes} \\ & = 76,275,253,248 \text{ bytes} \approx \textbf{76.28} \text{ GB} \end{aligned}$$

This makes storing the dataset with calculated priors impractical, be it in memory or disk, quantization allows us to cut these requirements down to a quarter of the original size.

Prior generation scripts were succesfully implemented for:

- Depth
- Surface Normals
- Segmentation

All prior generation scripts can be found in the `/geometry-priors` folder within the 3DGS-priors repository.

As to improve training and evaluation performance, we chose to generate priors in advance for selected datasets. As such we developed a pipeline that executes the prior generation scripts and constructs a ready-to-use dataset, alongside a custom DataLoader that can read said dataset. Implementation details can be found in **Section 2.4: Data pipelines**.

2.3.5 Model Changes

The first change to the model was to have the top-level `GaussianSplatPredictor` class dynamically calculate the number of input channels required based on the training configuration, this information was then passed to the underlying Convolutional layers and UNet blocks during initialisation.

```
def calc_channels(cfg):
    # Base RGB channels
```

```
    in_channels = 3

    # Older configs may not have relvant options, select() returns None if the option is
    missing
    if OmegaConf.select(cfg, "data.use_pred_depth") is True:
        in_channels += 1
    if OmegaConf.select(cfg, "data.use_pred_normal") is True:
        in_channels += 3

    return in_channels
```

Listing 2: Channel calculation code

```
# Calculate number of input channels
self.in_channels = calc_channels(cfg)

# Initialise correct model depending on if Gaussian mean offsets are to be calculated
if cfg.model.network_with_offset:
    split_dimensions, scale_inits, bias_inits = self.get_splits_and_inits(True, cfg)
    self.network_with_offset = networkCallBack(cfg,
                                cfg.model.name,
                                self.in_channels,
                                split_dimensions,
                                scale = scale_inits,
                                bias = bias_inits)
    assert not cfg.model.network_without_offset, "Can only have one network"
if cfg.model.network_without_offset:
    split_dimensions, scale_inits, bias_inits = self.get_splits_and_inits(False, cfg)
    self.network_wo_offset = networkCallBack(cfg,
                                cfg.model.name,
                                split_dimensions,
                                scale = scale_inits,
                                bias = bias_inits)
    assert not cfg.model.network_with_offset, "Can only have one network"
```

Listing 3: New model initialisation code

added film layer for text input

One of our desired features was to allow a model with new priors to be fine-tuned using existing weights, namely those from the pretrained Splatter Image models. This was achieved by grafting the old weights of a pretrained model onto a newly generated GaussianSplatPredictor. This grafting mechanism was implemented externally to the model, as such its implementation details are covered in **Section 2.3.6: Training and Evaluation Changes**.

2.3.6 Manual LoRA Integration

To adapt the Splatter Image architecture for fine-tuning on the ShapeNet-SRN Cars dataset (with depth and normal priors), we added Low-Rank Adaptation (LoRA) [17] into the GaussianPreidictor module of the Splatter Image. LoRA adds low-rank trainable matrices into frozen, pretrained layers. This preserves the base model, while further finetuning it to the new data.
Existing libraries such as Hugging Face's PEFT [37] provide out-of-the-box LoRA integration, we found them incompatible with the weight-grafting mechanism required by the SplatterImage pipeline. Splatter Image relies on grafting weights from pre-trained checkpoints where input channel dimensions change (e.g., when adding depth or normal map channels). This leads to shape mismatches during initialisation steps (such as for PEFT's LoRA layers) since the pre-trained weights cannot be directly loaded into layers with modified input dimensions.
We added LoRA manually within the `GaussianPredictor` and `train_network` modules, using code from Microsoft's `loralib` [18]. We modified the underlying `Linear` and `Conv2d` layers of the U-Net architecture

to include a `LoRALayer` mixin. This allows the freezing of pre-trained base weights while injecting the LoRA matrices (A and B) directly into the forward pass. As mentioned earlier, this preserves the frozen pretrained weights from the original 3-channel RGB model, but is compatible with the new channels.

2.3.7 Training and Evaluation Changes

The existing Splatter Image evaluation script needed minimal changes, thanks to how we integrated our priors into the project. The training script initially needed minimal changes, but then was split into two versions, one with minimal changes for standard training, and one with additional changes required for LoRA support.

For evaluation code, the first change was adding support for loading our pretrained models available on HuggingFace.

```
# Load pretrained model from HuggingFace if no local model specified
if args.experiment_path is None:
    # Eval run on the our new dataset with priors
    if dataset_name in ["cars_priors"]:
        cfg_path = hf_hub_download(repo_id="MVP-Group-Project/splatter-image-priors",
                                   filename="model-depth-normal/config.yaml")
        model_path = hf_hub_download(repo_id="MVP-Group-Project/splatter-image-priors",
                                filename="model-depth-normal/model_best.pth")

    # Eval run on previous Splatter Image datasets
    else:
        cfg_path = hf_hub_download(repo_id="szymanowiczs/splatter-image-v1",
                                   filename="config_{}.yaml".format(dataset_name))
        if dataset_name in ["gso", "objaverse"]:
            model_name = "latest"
        else:
            model_name = dataset_name
        model_path = hf_hub_download(repo_id="szymanowiczs/splatter-image-v1",
                                filename="model_{}.pth".format(model_name))

else:
    cfg_path = os.path.join(experiment_path, ".hydra", "config.yaml")
    model_path = os.path.join(experiment_path, "model_latest.pth")

# load cfg
training_cfg = OmegaConf.load(cfg_path)
```

Listing 4: New model loading code

Input preparation was also changed, namely priors are concatenated (if enabled) as additional channels to the input RGB images, before being fed into the network. Splatter Image uses a DataLoader to receives batches of RGB images to perform infrfence on. The DataLoader does not directly return a tensor of images, but a dictionary containing relevant batch data, namely XXX XXX XX. This allowed our priors to be introduced into the preparation code easily, by simply providing them a key/value pairs in the dictionary provided by the DataLoader, in the form of tensors matching the RGB image batch. The priors can be accessed by KERY AND then concatenated with RGB images in a specific order as to generate the input tensors for the model to perform inference on. Assertions are also performed to verify that priors are indeed available for the currently active dataset.

```
# Concatenate selected priors
input_images = data["gt_images"][:, :model_cfg.data.input_images, ...]
if "use_pred_depth" in model_cfg.data and model_cfg.data.use_pred_depth:
    assert model_cfg.data.category == "cars_priors", "Dataset does not have predicated maps!
    "
    input_images = torch.cat([input_images,
                    data["pred_depths"][:, :model_cfg.data.input_images, ...]],
                    dim=2)
```

```
if "use_pred_normal" in model_cfg.data and model_cfg.data.use_pred_normal:
    assert model_cfg.data.category == "cars_priors", "Dataset does not have predicated maps!
    "
    input_images = torch.cat([input_images,
                    data["pred_normals"][:, :model_cfg.data.input_images, ...]],
                    dim=2)

# Get camera to center depth
if model_cfg.data.origin_distances:
    input_images = torch.cat([input_images,
                    data["origin_distances"][:, :model_cfg.data.input_images, ...]],
                    dim=2)
```

Listing 5: Network input preparation code

For the training code, .

ADDED HF LAODING, PERFORMS GRAFT

ADDED CONATENATION¡ DATASET LAODING REMAINS THE SAME

For the LoRA-specific training pipeline, within the modified `GaussianPredictor`, `requires_grad=False` is explicitly set for all pre-trained backbone weights (see `gaussian_predictor_lora.py`), leaving only the low-rank matrices A and B as trainable. This ensures that optimiser updates are restricted to the adapter layers, with instances defined as `LoRALayer`.

2.4 Data pipelines

[Explain your data format, how you consume the data in your algorithms, and data augmentation.]

DIAGRAM: SRN FILE STRUCTURE -¿ ORCHESTRATOR -¿ PARALLEL MODELS -¿ PARQUET DIAGRAM: PARQUET -¿ HUGGINGFACE STRUCTURE -¿ HUGGINGFACE

For training and evaluation performance reasons, we chose to generate priors in advance for selected datasets, as such we developed a pipeline, run by an orchestrator pattern notebook, that would set up a Virtual Machine for every model, then proceed to install the correct depednecies into each VM, using requiremtns files stored alongside the scripts in `/geometry-priors`, and then execute the prior generation notebooks in order to geenrate a complete, ready-to-use dataset. CODE BLOCK - VM setup CODE BLOCK - REQUIREMTN LOADING The parquet files can also be processed into a file structure appropriate for upload to HuggingFace, where the pregenereated datset remains avaiable. Models use the pregenerated datasets via a custom laoder we implemented, taht slots in into all of the exisitng training/evlaution code.

TO SPEED UP TRAINIG AND EVUALTION, WE PREGENERATE THE PRIORS FOR ALL MODELS USING THE MODELS SELECTED AS PER OUR research DOING SO WE GENERATE A NEW DATASET

FIRST THE USER DOWNLAODS AND SETS UP THE APPRIPORAITE DATASET AS PER SPLATTER IMAGE DOC RUNS GEENRTE DATASET NTOEBOOK WITH CORRECT ARGS, THIS WILL FOLLW ORDCHESTOR PTTERN DESCRIBED IN EARIELR SECTION MODELS OUTPUT PARQUET FIELS IN THE SET DESTIATION, WITH EXAMPLE ROWS: EXAMPLE RGBS EXAMPLE PSOES EXAMPLE INTRINS EXMAPLE DEPTHS EXAMPE NORMALS These can now be used however the user wants to, we add an additional processing step to transform aprquet files into file structure appropriate for uplaod to huggingface, matching the following config:

YAML CONFIG

A DIAGRAM OF HOW THIS WHOLE PEIPLEINE WORKS IS SEEN IN FIGURES

Take a concretne example, we offer one ready pregenerated dataset on HF: We took our input data from the ShapeNet-SRN dataset from [38] at 128×128 resolution. transform pipeline standard, the depth, rgb, normal columns store images as uint8s in C H W format, this is as there is X images, do math, thus this manmy GB of data, we must un fortunately for compute reasons take this loss of accuracy.

TRAINIGN LOADS HF DATASET LOAD HF PRETRAINED WEIGHTS PERFORMS WEIGHT GRAFT TRAINING OCCURS WANDB SAVES WEIGHT OCCASNIONALLY WEIGHTS ARE FUSED AT SAVE THESE WEIGHTS ARE UPLOADED TO HF WITH THIS NAMING SCHEME:

2.5 Training procedures

[Explain which framework and optimizers you use, how you implemented the training logic.]

THE ENTIRE MODEL IS PROGRAMMED USING TORCH WE USE PEFT OR WHATEVRE TO ADD LORA Layers

OUR TRAINING IS SET BY CONFIG FROM WADB?

SHWO CONFGI

WHEN MODEL TRAINIGN OCCURS CONFIG OPTIOSN ARE VERFIEID TO BE COMPATIABLE OPTIONS INCLUIDE USING LORA vs NOT, WHICH PRIORS TO USE, AND WHETEHR YOU WNAT TO START THE MODEL WITH EXISITNG WEIGHTS (if so grafted) WEIGHTS FUSED AT SAVE

TRAINGING IS RUN USING DEFAULT SPALTTER IAMGE APRAMSTERS, Namely: LIST OPTIMSIER OtHER OPTIOSM

THE AVAIABLE MODCEL WEIGHST ON HF WERE TRAINIG NNTOEBOOK RUSN ON CLOAB WITH A100 ITERS: 60K

In order to isolate the parameter-efficient adaptation due to LoRA, we used a dedicated training script (`train_network_lora.py`). While preserving the data pipeline and training loop architecture as the standard framework, it is necessary to enforce strict freezing where the gradients for the U-Net backbone are disabled. Optimisation is therefore restricted solely to the injected low-rank matrices (as described in **2.2.1 Low-Rank Adaptation (LoRA)**).

2.6 Testing and validation procedures

[Explain which framework you use, how you implemented the testing/ validation logic.]

Chapter 3: Experiments and Evaluation

3.1 Datasets

[Explain the datasets utilized: what they contain, why they are utilized, assumptions, limitations, possible extensions.]

The standard benchmark for evaluating single-view 3D reconstruction is ShapeNet-SRN [38], hence we used this to test and evaluate our main model implementation. For this dataset, we specifically use the "Car" class, which used the "car" class of ShapeNet v2 [20] with 2.5k 3D CAD model instances. The SRN dataset was generated by disabling transparencies and specularities and training on 50 observations of each instance at a resolution of 128×128 pixels, with camera poses being randomly generated on a sphere with the object at the origin. A limitation of this dataset is the lack of subject variety in the dataset as the model may end up overfitting to cars. A possible extension to address this limitation could be to include other classes in the ShapeNet-SRN database to make sure that the model can still generalise to other types of objects.

An extension of this dataset is implemented in [19], which presents a Deep Implicit Surface Network to generate a 3D mesh from a 2D image by predicting the underlying signed distance fields. In the paper, they generated a 2D dataset composed of renderings of the models in ShapeNet Core [20]. For each mesh model, the dataset provides 36 renderings with smaller variation and 36 views with larger variation (bigger yaw angle range and larger distance variation). The object is allowed to move away from the origin, which provides more degrees of freedom in terms of camera parameters, and the "roll" angle of the camera is ignored since it was deemed very rare in real-world scenarios. The images were rendered at a higher resolution of 224×224 pixels and were paired with a depth image, a normal map and an albedo image as shown in figure 5. This dataset was mainly used as a ground truth to evaluate the generation of geometry priors (e.g. normal map and depth map). A limitation of this dataset would be its small size since only 72 samples are available for us to use, such that the performance of geometry prior generation may not be evaluated correctly. However, in the same GitHub repository, the script to generate these images from the ShapeNet Core dataset is provided, so a possible extension given more time could be to include more images by running the script on other objects in the ShapeNet Core dataset.

3.2 Training and testing results

[Explain the training and testing results with graphs and elaborating on why they make sense, what could be improved.]

3.3 Qualitative results

[Show in figures and explain visual results. Include different interesting cases covering different aspects/ limitations/ dataset diversity. If not converged, explain what we can expect once converged. Include any other didactic examples here.]

3.4 [Optional] Quantitative results

[A table and associated explanations for quantitative results.]

3.4.1 LoRA experiment results

To evaluate whether Low-Rank Adaptation (as described in **Section 2.3.5: Model Changes**) is effective, we performed a set of finetuning experiments. LoRA behaviour can be adjusted through three hyperparameters: (LINK BACK TO THE SUBSECTIONS IN SECTION 2 HERE)

We tested multiple configurations by varying these hyperparameters, while keeping all other components. We measured the performance using PSNR, SSIM and LPIPS.

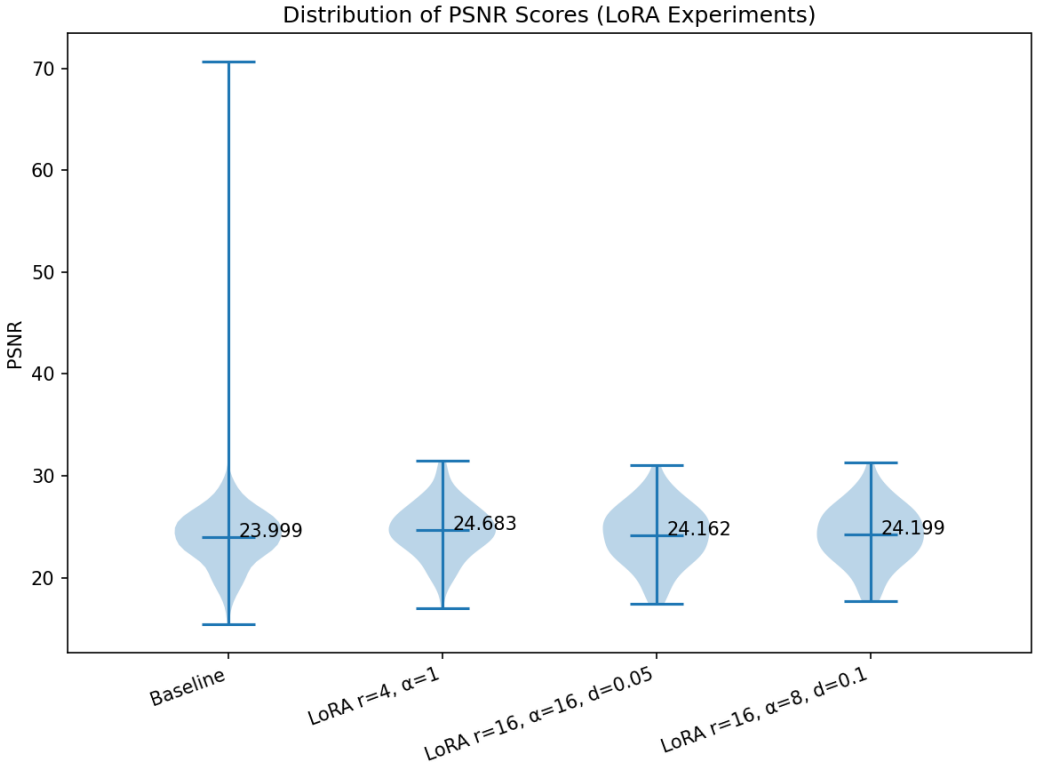


Figure 14: PSNR distribution for different LoRA configurations

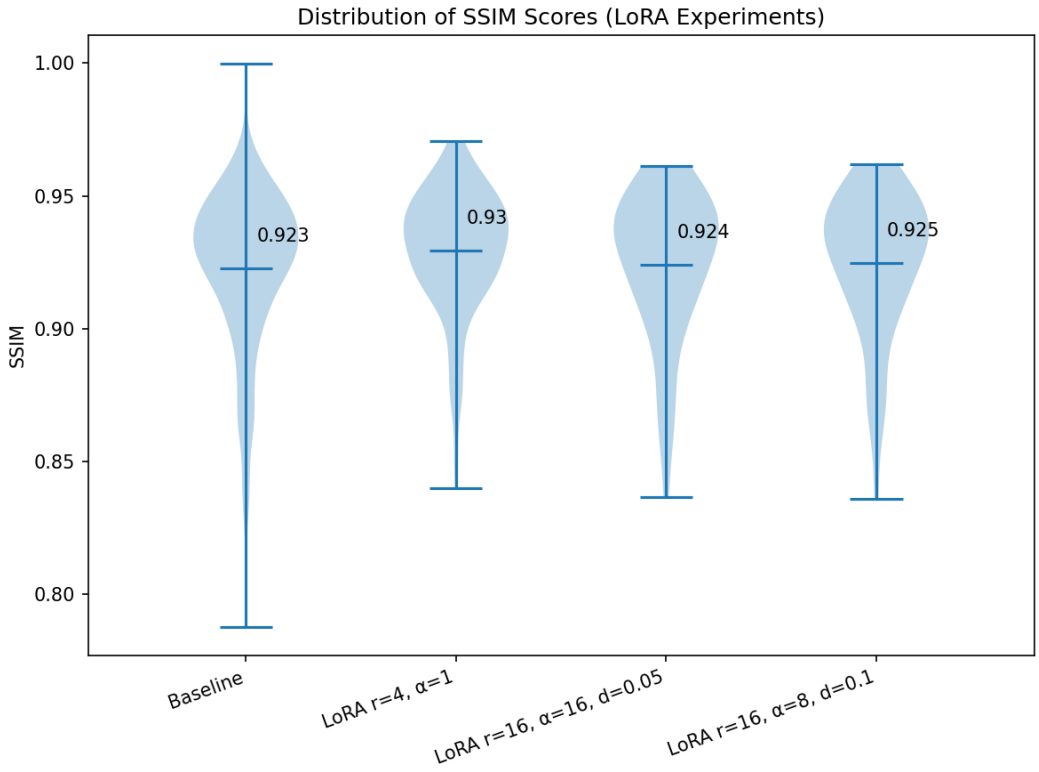


Figure 15: SSIM distribution for different LoRA configurations

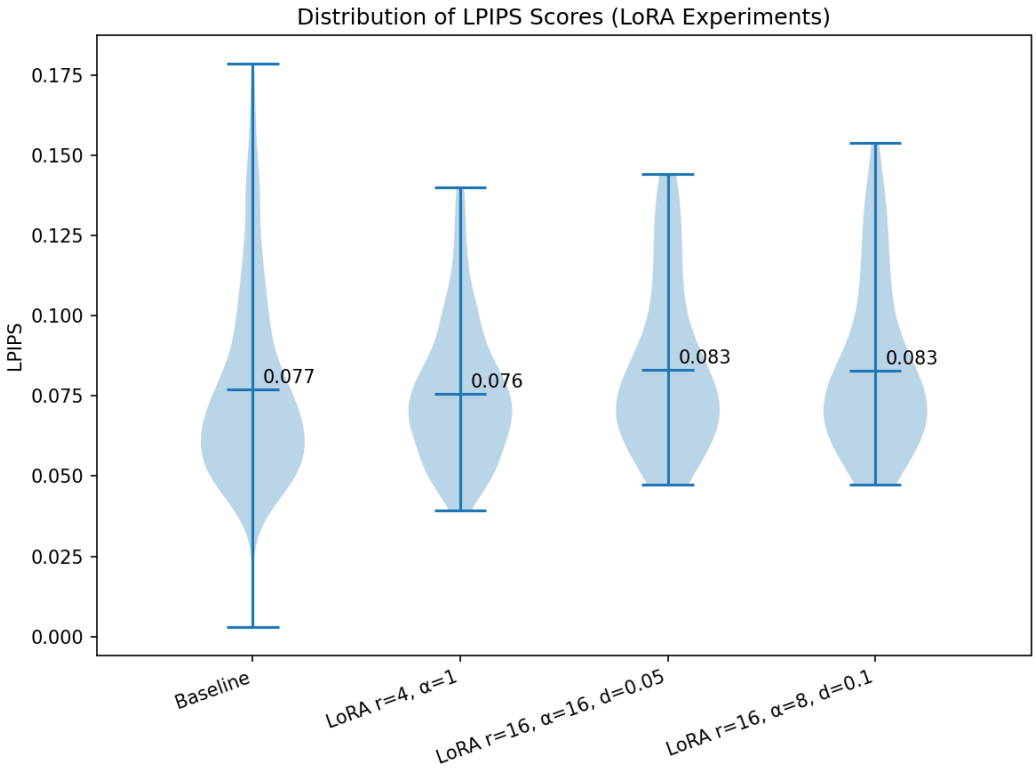


Figure 16: LPIPS distribution for different LoRA configurations

3.5 [Optional] Comparison to state-of-the-art

[Qualitative and/ or quantitative comparisons to one or more recent works, especially the baseline work.]

(Placeholder LoRA results comparison with SplatterImage) We can compare our LoRA results with the original Splatter Image paper [4].

Chapter 4: Conclusions and Future Directions

4.1 Conclusions

[Summarize what the project was about and the main conclusions.]

4.2 Discussion of limitations

[Explain the limitations of your technique. You may want to refer to previous sections or show figures on the limitations.]

halluciantion in hidden areas still a problem, as at end of the day you dont know what there, only doign best guess basd on alrge datasets lot of data cnd comptue needed to perofrm accurate reweults we lacked compute to generate and test more models priors

4.3 Future directions

[State a few future directions for research and development. These typically follow from the discussion on limitations.]

generating and testing mroe priors, trying longer training times, largert datasets more priors aoriented to hidden areas, could bring up planes again,

4.4 Project Contributions

"You may find the template for the project report here. We do not enforce any page limits but please make sure to address each section appropriately as explained in the document. In particular, please pay special attention to clarifying the contribution of each group member." Should we clarify here or throughout document?

Section 1.1: Kacper and Alex Section 1.2: Kacper Section 1.3: Kacper Section 2.1: Kacper and Radhika and Alex Section 2.2: Kacper and Radhika Section 2.3: Kacper and Radhika and Alex Section 2.4: Kacper Section 2.5: Kacper and Radhika Section 2.6: Kacper Section 3.1: FILL IN Section 3.2: Radhika Section 3.3: FILL IN Section 3.4: FILL IN Section 3.5: FILL IN Section 4.1: FILL IN Section 4.2: FILL IN Section 4.3: FILL IN

Depths exploration: Radhika Segmentation exploration: Radhika Normals exploration: Alex Planes exploration: Alex

Splatter Image setup and bugfixes: Radhika and Alex Splatter Image architectural modification (Grafting, Channel changes, FiLM, Cross-Attention): Kacper Splatter Image LoRA integration: Radhika Splatter Image Training Modification: Kacper Splatter Image Eval Modification: Radhika and Kacper generate depths: Kacper generate normals: Kacper generate segmentation: Radhika Splatter Image cars priors dataloader: Kacper HF dataset orchestorator: Kacper

Testing dataloader: Kacper Testing model: Kacper Testing LoRA configurations: Radhika

Results processing/graphing code: Alex Image collection, citation collection and verification: Alex

References

[1] 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, 2020.

[2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023.

[3] Shengji Tang, Weicai Ye, Peng Ye, Weihao Lin, Yang Zhou, Tao Chen, and Wanli Ouyang. Hisplat: Hierarchical 3d gaussian splatting for generalizable sparse-view reconstruction, 2024.

[4] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Splatter image: Ultra-fast single-view 3d reconstruction, 2024.

[5] Jianghao Shen, Nan Xue, and Tianfu Wu. A pixel is worth more than one 3d gaussians in single-view 3d reconstruction, 2024.

[6] Zhizhong Kang, Juntao Yang, Zhou Yang, and Sai Cheng. A review of techniques for 3d reconstruction of indoor environments. *ISPRS International Journal of Geo-Information*, 9(5), 2020.

[7] Andrea Romanoni, Amaël Delaunoy, Marc Pollefeys, and Matteo Matteucci. Automatic 3d reconstruction of manifold meshes via delaunay triangulation and mesh sweeping, 2016.

[8] Timothy Newman and Hong Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30:854–879, 10 2006.

[9] Yuxin Wang, Qianyi Wu, and Dan Xu. F3d-gaus: Feed-forward 3d-aware generation on imagenet with cycle-aggregative gaussian splatting, 2025.

[10] Junlin Hao, Peiheng Wang, Haoyang Wang, Xinggong Zhang, and Zongming Guo. Gaussvideodreamer: 3d scene generation with video diffusion and inconsistency-aware gaussian splatting, 2025.

[11] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks, 2022.

[12] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-xl: A universe of 10m+ 3d objects, 2023.

[13] 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, 2022.

[14] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. Lrm: Large reconstruction model for single image to 3d, 2024.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[16] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[17] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[19] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *NeurIPS*, 2019.

[20] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[21] Gwangbin Bae, Ignas Budvytis, and Roberto Cipolla. Estimating and exploiting the aleatoric uncertainty in surface normal estimation. In *International Conference on Computer Vision (ICCV)*, 2021.

[22] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[23] Jingwei Huang, Yichao Zhou, Thomas Funkhouser, and Leonidas Guibas. Framenet: Learning local canonical frames of 3d surfaces from a single rgb image. *arXiv preprint arXiv:1903.12305*, 2019.

[24] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[25] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 283–291, 2018.

[26] Xiaojuan Qi, Zhengzhe Liu, Renjie Liao, Philip HS Torr, Raquel Urtasun, and Jiaya Jia. Geonet++: Iterative geometric neural network with edge-aware refinement for joint depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[27] Saeed Mahmoudpour and Manbae Kim. Chapter 10 - a study on the relationship between depth map quality and stereoscopic image quality using upsampled depth maps. In Leonidas Deligiannidis and Hamid R. Arabnia, editors, *Emerging Trends in Image Processing, Computer Vision and Pattern Recognition*, pages 149–160. Morgan Kaufmann, Boston, 2015.

[28] Xinwei Liu, Marius Pedersen, and Renfang Wang. Survey of natural image enhancement techniques: Classification, evaluation, challenges, and perspectives. *Digital Signal Processing*, 127:103547, 2022.

[29] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[30] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[31] Daniel Gatis. rembg. `https://github.com/danielgatis/rembg`, 2025. Version 2.0.66.

[32] Taehun Kim, Kunhee Kim, Joonyeong Lee, Dongmin Cha, Jiho Lee, and Daijin Kim. Revisiting image pyramid structure for high resolution salient object detection. In *Proceedings of the Asian Conference on Computer Vision*, pages 108–124, 2022.

[33] Peng Zheng, Dehong Gao, Deng-Ping Fan, Li Liu, Jorma Laaksonen, Wanli Ouyang, and Nicu Sebe. Bilateral reference for high-resolution dichotomous image segmentation. *CAAI Artificial Intelligence Research*, 3:9150038, 2024.

[34] Radhakrishna Achanta, Sheila Hemami, Francisco Estrada, and Sabine Susstrunk. Frequency-tuned salient region detection. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1597–1604, 2009.

[35] Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu. Global contrast based salient region detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):569–582, 2015.

[36] Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A benchmark. *IEEE Transactions on Image Processing*, 24(12):5706–5722, December 2015.

[37] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

[38] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations, 2020.