

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 whilst maintaining high quality reconstruction, thereby improving computational efficiency and reducing data acquisition hardware requirements. Namely, in 2020 Neural Radiance Fields (NeRF) [1] introduced a revolutionary method for 3D reconstruction capable of high quality novel view synthesis. This is done by learning a continuous volumetric density and radiance function, typically using a deep learning model, which can then be queried by a ray-march for some new camera pose, enabling novel views to be synthesized. In 2023, 3D Gaussian Splatting (3DGS) [2] introduced an alternative method, offering major improvements in computational performance. Instead of an implicit function, 3DGS introduces a new explicit representation, that of a set of 3D Gaussians (called a Gaussian splat), Gaussians are volumes defined by parameters such as position, colour, opacity, rotation and scale, which can be efficiently rasterized to generate novel views. Reconstruction is typically achieved by optimizing the set of Gaussians to produce novel view with minimized loss, alternatively deep learning methods can be used to directly predict the Gaussian splat. 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].

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 has 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. We believe there are two reasons for this problem. The first is inherent to Splatter Images architecture, unlike methods that utilize explicit 3D feature volumes, Splatter Image’s choice of 3D reconstruction as a 2D-to-2D image translation task limits its ability to learn geometric priors, as the model lacks an internal 3D representation to resolve issues like depth ambiguities. The second is that the 3DGS prediction based on only single or few RGB image features alone does not have sufficient conditional information for Splatter Image to infer appropriate geometry information or 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.

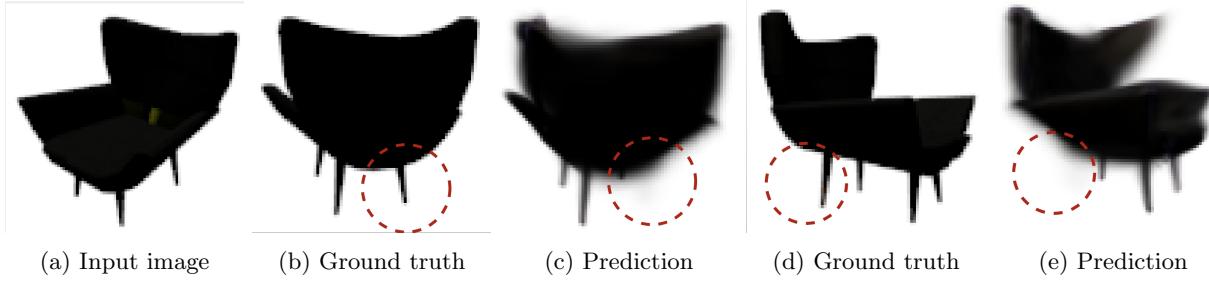
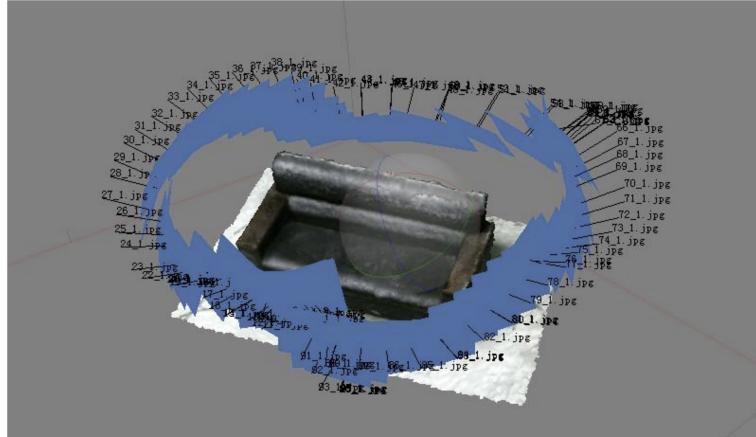


Figure 1: Splatter Image outputs compared with ground truth taken from [5]

This project aims to address these issues, improving reconstruction quality, 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, by providing necessary additional information and preventing Splatter Image from having to learn how to generate 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.]



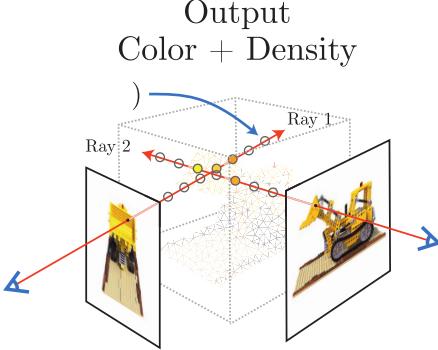


Figure 3: NeRF implicit function from [1]

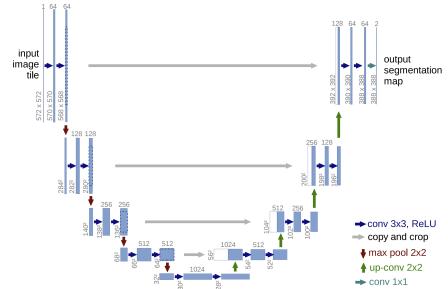


Figure 4: 3D Gaussian Splatting Process from [2]

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, 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 rendered 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 VIDEOODREAMER, 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 TODO

Gauss VideoDreamer TODO

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]. TODO

Another recent method is Splatter Image [4] then applies Gaussian Splatting to monocular reconstruction 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. TODO

1.3 Overview of the idea

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

Splatter Images high computational performance and relative reconstruction quality makes it a highly desirable model for one-shot 3D reconstruction

the model still occasionally suffers from challenges such as layout/scale drift, over-smooth geometry and poor quality hallucinations in occluded regions, particularly in reconstructing structures unseen in the input view, including for views significantly different from the source.

We believe there are two reasons for this problem. The first is inherent to Splatter Images architecture, unlike methods that utilize explicit 3D feature volumes, Splatter Image's choice of 3D reconstruction as a 2D-to-2D image translation task limits its ability to learn geometric priors, as the model lacks an internal 3D representation to resolve issues like depth ambiguities. The second is that the 3DGS prediction based on only single or few RGB image features alone does not have sufficient conditional information for Splatter Image to infer appropriate geometry information or structures that are not visible in the input view.

this is fundamanetally due to the lack of inforamtion the model has from a single input image

Currently there exists specialised, accurate models that can perorm on-shot prediction with hig haccuracy specific geomtric features of images trained on many examples so store lot of konwdleg about corectly halkucinating occluded reegions in a realstic/more accuraet manner

We propsoe exploiting these models to gain additional sources of inormation which could be fed into the model these priros solve both of our beleived problems, namely they provide provide large amounts of additional information, tackling issue no 2, and stop the model from nedign to learn hwo to infer these geometric features itself, which it is unsuitable/poor at, tackling issue no 1. We bleive as such these priors would guide recosntruction in a more accurate manner, impriving the recosntruction quality of the model, whislt requring minimal architectural changes to Splatter Image, as only have to change inputu channel count and do minor modifications if we want to add multimodal data, generally preserving its performance,

we also propose performing ablation study to see which priors are most effective/significant in changing the reconstruction quality

THJERBY TACKLING

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.]

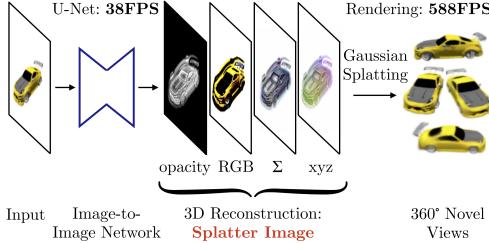


Figure 5: Overview of SplatterImage[4]

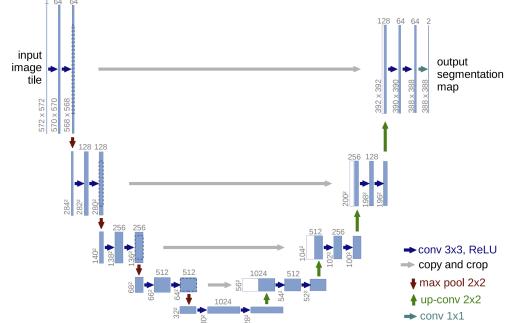


Figure 6: U-net architecture[15] that Song U-Net[16] is based on

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, 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.]

Our first modification is to allow the Splatter Image model to dynamically calculate and initialise the neccesary number of input laeyrs based on the supplied model configuration. This will allow it to create models that use the desired combination of additional priors.

The second addition is of FiLM layers to support multimodal input. CERTAIN PRIORS ARE NOT MAPS, SAY Planes or SEGEMTNAION TEXT TOKESN THESE CAN BE SUBMITTED TO THE MODEL VIA METHDOS LIKE BROADCASTIGN WHERE YOU CAST THE same VALUE FOR EACH PXIERL AND ADD A CHANNEL, THIS LIMITED BY NUMBER OF KNOWN Layers AND IS HIGHYL INEFFICIENT TOHIGH ALTERNATIVE METHOD TO SUPPLY MULTIMODAL DATA IS TO MODULE LAYER-S/CHANNELS BEWTEEN the U-NET/coNVolitonal blocks in the model, via the use of FiLM layers. EXPLAIN FILM 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: $y = \text{gamma} * x + \text{beta}$. This modulated feature map is passed to the

next layer of the U-Net. Existing U-Blocks support FiLM for camera embeddings, we modify U-net blocks to have additional film layer for segmentation tokens

An alternative to FiLM is also considered, namely cross-attention. Also new cross-attention modules (slot in replacement for FiLM) 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 considered state-of-the-art for high-performance conditional generation.

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:

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

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 initialisation, 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 considered providing the model with structural information to be one of the most likely avenues of improvement. These structural priors were considered in two flavours, in the form of predicted scene planes and scene surface normal maps.

When researching plane prediction, we reached the conclusion that this flavour would in fact be unlikely to help guide reconstruction. For example teddy bears (as seen on the CO3D Teddybears dataset) have complex, convex shapes, lacking dominant planes on their surface, using a planar prior might confuse the network, causing it to flatten the bear’s features or causing poor quality hallucination; 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 the orientation of a surface on a per-pixel level, we considered this to be an excellent prior as it supports both complex shapes such as teddy bears, but can equally well describe planar surfaces. Hence, we selected this prior as a prime candidate that could improve the models 3D surface reconstruction and help guide accurate hallucination in occluded regions. For our ground truth we use [19] which provides a dataset of higher resolution images of ShapeNet models from [20], each RGB image is paired with a depth map, a normal map and an albedo map at https://github.com/Xharlie/ShapenetRender_more_variation. We feed these images into the normal map generation models and compare against the ground truth normal maps, using Pixel Based Visual Information Fidelity as a metric to evaluate their performance.

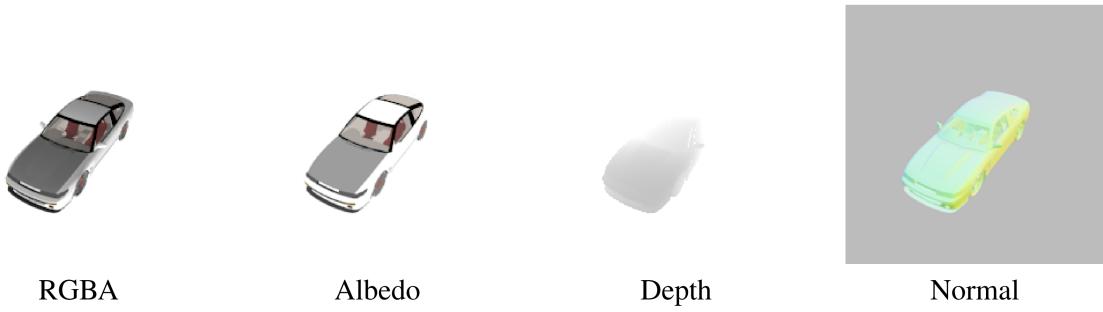


Figure 7: Example of image with maps used as ground truth taken from [21]

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.

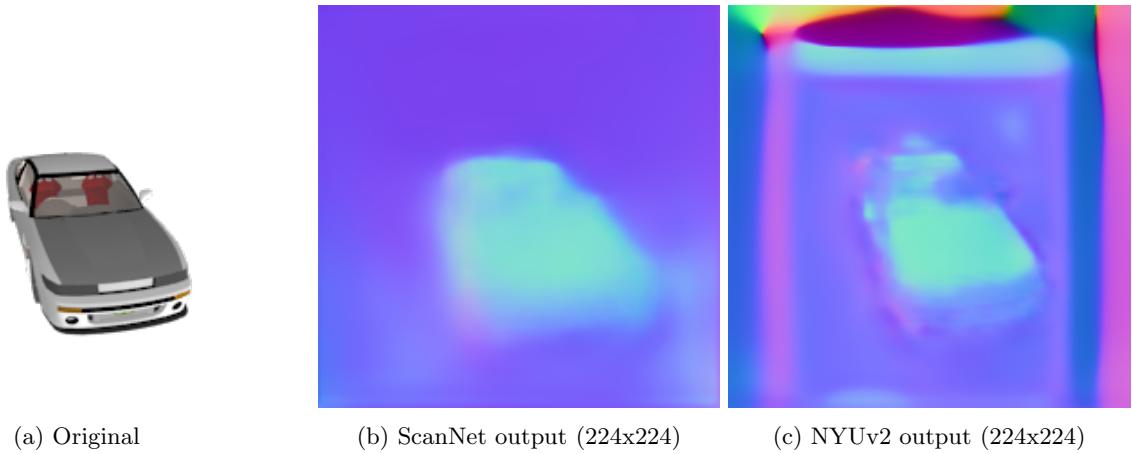


Figure 8: Comparison of original input and two model outputs

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.

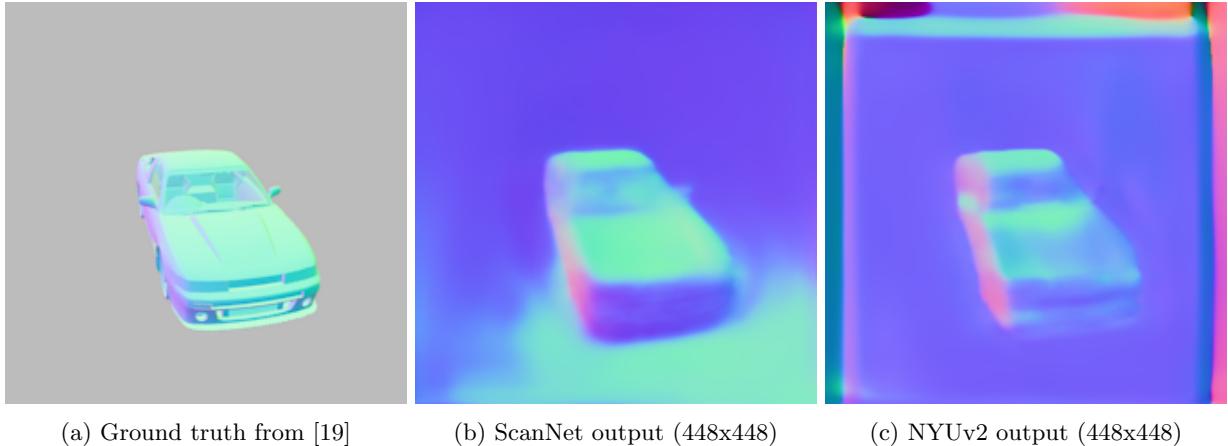
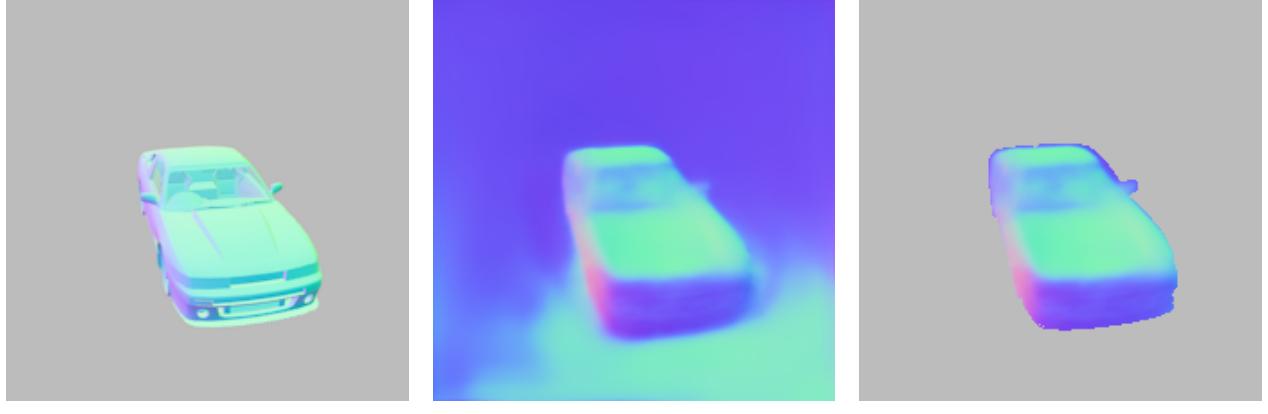


Figure 9: Comparison of model outputs when setting input dimensions as 448x448 instead of 224x224 alongside ground truth

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 8c and 9c 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 10: 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].

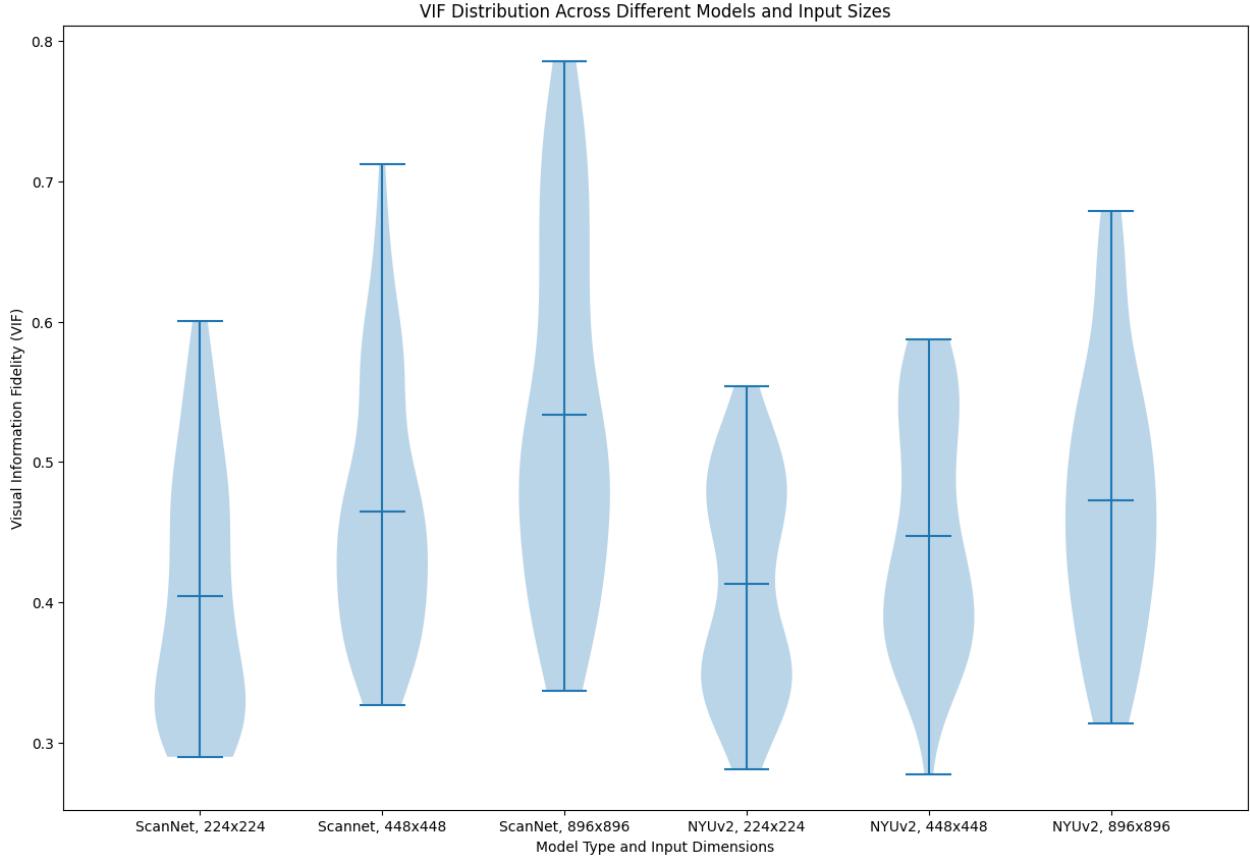


Figure 11: Comparison of VIF between ground truth and different models

From Figure 11 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].

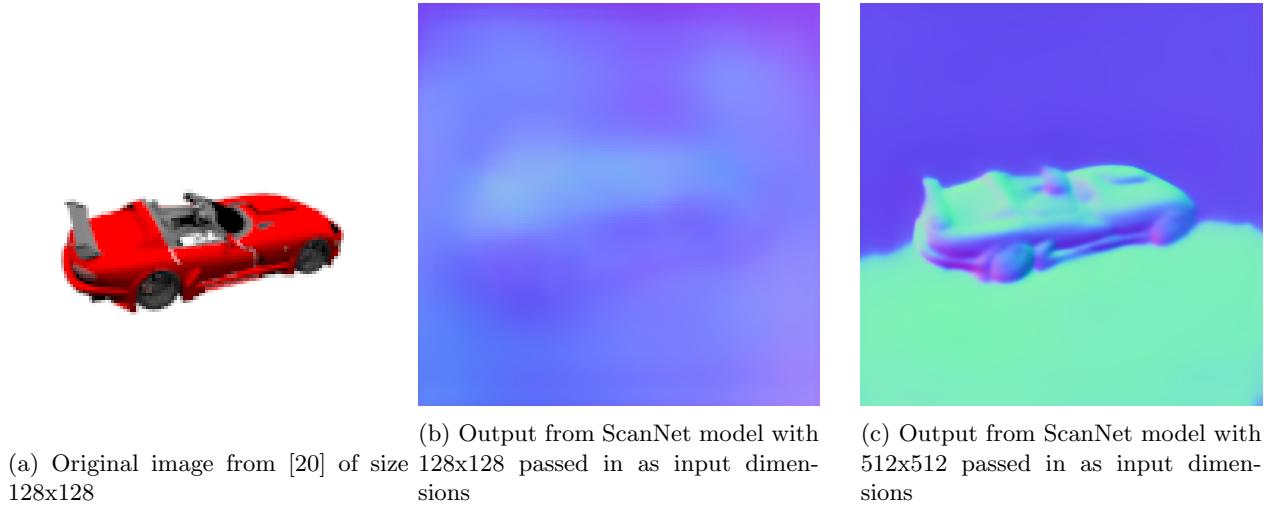


Figure 12: Original ShapeNet image and normal map outputs

Without passing in dimensions larger than the input image into the model, we can see from comparing Figures 8a and 8b to Figures 12a and 12b 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 12c.

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 7. 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 summarises the metrics across the MiDaS models tested.

Table 1: Comparison of MiDaS models on set of easy and hard images.

Difficulty	Model	AbsRel ↓	RMSE ↓	SI-RMSE ↓	$\delta < 1.25 \uparrow$
Easy	DPT_Hybrid	0.089 ± 0.12	20.38 ± 19.39	0.123	0.909
Easy	DPT_Large	0.091 ± 0.12	20.56 ± 19.74	0.124	0.909
Easy	MiDaS_small	0.096 ± 0.13	21.54 ± 20.44	0.129	0.918
Hard	DPT_Hybrid	0.101 ± 0.15	19.65 ± 17.99	0.128	0.907
Hard	DPT_Large	0.170 ± 0.41	22.40 ± 22.12	0.151	0.906
Hard	MiDaS_small	0.190 ± 0.45	24.45 ± 23.56	0.164	0.900

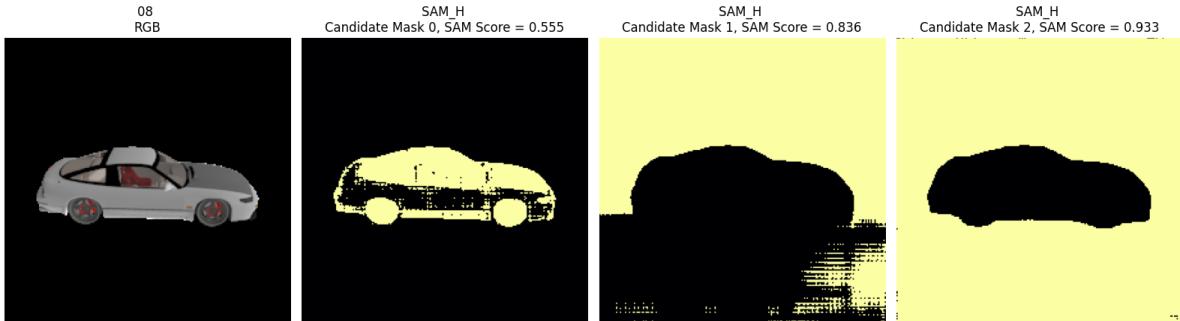
The quantitative degradation of DPT_Large on the Hard set contradicts visual inspection. This discrepancy can be attributed to the quality of the available Ground Truth (GT) depth maps (as discussed earlier). DPT_Large is what is used in the final depth prior generation, as described in **2.3.4 Selected Prior Integration**.

INSERT EXAMPLE IMAGE HERE OF DEPTH MAP PRODUCED

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 13, 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 13: 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:

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

It is commonly used in 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 2: Comparison of Salient Object Detection models on ShapeNet renders. \uparrow indicates higher is better, \downarrow indicates lower is better.

Difficulty	Model	IoU \uparrow	$F_\beta \uparrow$	MAE \downarrow
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 14 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.

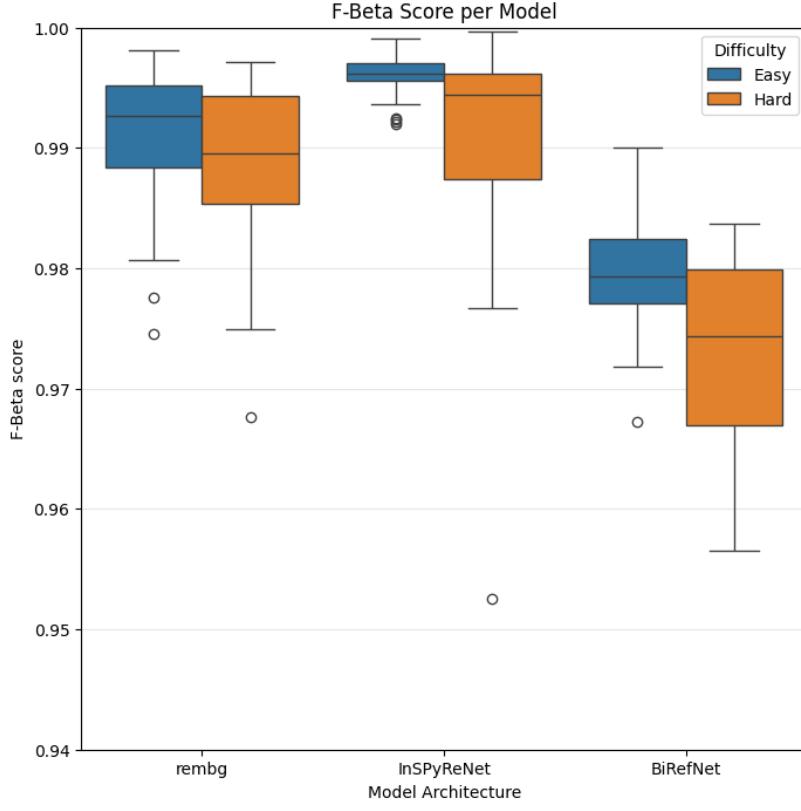


Figure 14: Distribution of F_β scores for each model across the two difficulty levels.

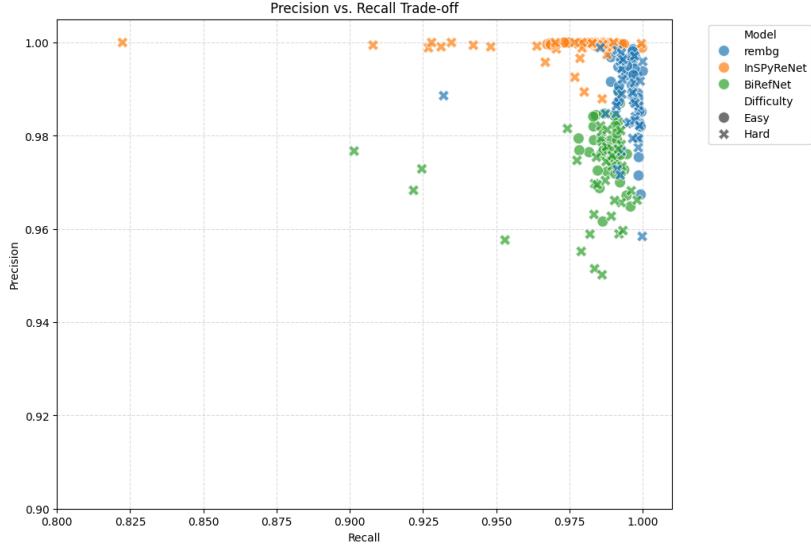


Figure 15: Precision vs. Recall trade-off

The scatter plot in Figure 15 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. InSpReNet 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.

```

1 with torch.no_grad():
2     for batch_imgs, batch_filenames in loader:
3         # Prediction, B C H W format
4         batch_imgs = batch_imgs.to(device)
5         preds = model(batch_imgs)
6
7         # Batched downsize
8         preds = torch.nn.functional.interpolate(
9             preds.unsqueeze(1),
10            size=target_size,
11            mode="bicubic",
12            align_corners=False
13        ).squeeze(1)
14
15         # Batched 0 to 1 normalization
16         batch_flat = preds.flatten(start_dim=1)
17         min_val, max_val = torch.aminmax(batch_flat, dim=1, keepdim=True)
18         min_val = min_val.view(preds.size(0), 1, 1)
19         max_val = max_val.view(preds.size(0), 1, 1)
20         preds_normalized = (preds - min_val) / (max_val - min_val + 1e-8)
21         preds_uint8 = preds_normalized.mul(255).byte().cpu().numpy()
22
23         # H W format -> 128 * 128
24         for j, file_id in enumerate(batch_filenames):
25             full_batch.append(ProcessedImage(
26                 uuid=uuid,
27                 file_id=file_id.item(),
28                 image=preds_uint8[j].tobytes()
29             ))

```

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:

Depths: $387,956 \times 128 \times 128 \times 4$ bytes
 $= 25,425,084,416$ bytes $\approx \mathbf{25.43}$ GB

Normals: $387,956 \times 128 \times 128 \times 3 \times 4$ bytes
 $= 76,275,253,248$ bytes $\approx \mathbf{76.28}$ GB

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 successfully 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.

```

1 def calc_channels(cfg):
2     # Base RGB channels
3     in_channels = 3
4
5     # Older configs may not have relevant options, select() returns None if the option is
6     # missing
7     if OmegaConf.select(cfg, "data.use_pred_depth") is True:
8         in_channels += 1
9     if OmegaConf.select(cfg, "data.use_pred_normal") is True:
10        in_channels += 3
11
12     return in_channels

```

Listing 2: Channel calculation code

```

1 # Calculate number of input channels
2 self.in_channels = calc_channels(cfg)
3
4 # Initialise correct model depending on if Gaussian mean offsets are to be calculated
5 if cfg.model.network_with_offset:
6     split_dimensions, scale_inits, bias_inits = self.get_splits_and_inits(True, cfg)
7     self.network_with_offset = networkCallBack(cfg,
8                                                 cfg.model.name,
9                                                 self.in_channels,
10                                                split_dimensions,
11                                                scale = scale_inits,
12                                                bias = bias_inits)

```

```

13     assert not cfg.model.network_without_offset, "Can only have one network"
14 if cfg.model.network_without_offset:
15     split_dimensions, scale_inits, bias_inits = self.get_splits_and_inits(False, cfg)
16     self.network_wo_offset = networkCallBack(cfg,
17         cfg.model.name,
18         split_dimensions,
19         scale = scale_inits,
20         bias = bias_inits)
21 assert not cfg.model.network_with_offset, "Can only have one network"

```

Listing 3: New model initialisation code

added film layer for text input

Due to time and compute constraints, combined with the selected segmentation model not producing classification or instance labels, we did not implement the cross-attention mechanism for multimodal data input into the model, leaving it as an option for future development.

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 new instance of a 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 Splatter Image for fine-tuning on the ShapeNet-SRN Cars dataset (with depth and normal priors), we added Low-Rank Adaptation (LoRA) [17] into the GaussianPredictor module of Splatter Image. LoRA adds low-rank trainable matrices into frozen, pretrained layers. This preserves the base model, while further fine-tuning 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 our SplatterImage training pipeline. Due to compute limitations, we rely on Splatter Image grafting weights from pretrained 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.

```

1 # Load pretrained model from HuggingFace if no local model specified
2 if args.experiment_path is None:
3     # Eval run on the our new dataset with priors
4     if dataset_name in ["cars_priors"]:
5         cfg_path = hf_hub_download(repo_id="MVP-Group-Project/splatter-image-priors",

```

```

6     filename="model-depth-normal/config.yaml")
7     model_path = hf_hub_download(repo_id="MVP-Group-Project/splatter-image-priors",
8                               filename="model-depth-normal/model_best.pth")
9
10    # Eval run on previous Splatter Image datasets
11    else:
12        cfg_path = hf_hub_download(repo_id="szymanowiczs/splatter-image-v1",
13                                    filename="config_{}.yaml".format(dataset_name))
14        if dataset_name in ["gso", "objaverse"]:
15            model_name = "latest"
16        else:
17            model_name = dataset_name
18        model_path = hf_hub_download(repo_id="szymanowiczs/splatter-image-v1",
19                                     filename="model_{}.pth".format(model_name))
20
21    else:
22        cfg_path = os.path.join(experiment_path, ".hydra", "config.yaml")
23        model_path = os.path.join(experiment_path, "model_latest.pth")
24
25 # load cfg
26 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 the PyTorch DataLoader alongside custom Dataset objects to load batches of RGB images to perform inference on. The Dataset objects do not directly return a tensor of images, but a dictionary containing relevant batch data:

```

1 images_and_camera_poses = {
2     "gt_images": self.all_rgbs[example_id][frame_idxs].clone(),
3     "world_view_transforms": self.all_world_view_transforms[example_id][frame_idxs],
4     "view_to_world_transforms": self.all_view_to_world_transforms[example_id][frame_idxs],
5     "full_proj_transforms": self.all_full_proj_transforms[example_id][frame_idxs],
6     "camera_centers": self.all_camera_centers[example_id][frame_idxs]
7 }
8
9 images_and_camera_poses = self.make_poses_relative_to_first(images_and_camera_poses)
10 images_and_camera_poses["source_cv2WT_quat"] = self.get_source_cv2WT(images_and_camera_poses
11                           ["view_to_world_transforms"])
12
13 return images_and_camera_poses

```

Listing 5: Excerpt of srn.py Dataset code

This allowed our priors to be introduced into the evaluation code with ease, by simply creating a new custom Dataset object `srn_priors.py` that provides priors as new key/value pairs in the returned dictionary. Priors were specifically returned as PyTorch tensors matching the RGB image batch. The priors can then be accessed and concatenated with the RGB images in a specific order to generate the final input tensor for Splatter Image to perform inference on. Assertions are also performed to verify that priors are indeed available for the currently active Dataset.

```

1 # Concatenate selected priors
2 input_images = data["gt_images"][:, :model_cfg.data.input_images, ...]
3 if "use_pred_depth" in model_cfg.data and model_cfg.data.use_pred_depth:
4     assert model_cfg.data.category == "cars_priors", "Dataset does not have predicated maps!
5
6     input_images = torch.cat([input_images,
7                             data["pred_depths"][:, :model_cfg.data.input_images, ...]], dim=2)
8 if "use_pred_normal" in model_cfg.data and model_cfg.data.use_pred_normal:
9     assert model_cfg.data.category == "cars_priors", "Dataset does not have predicated maps!
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
305
306
307
307
308
309
309
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
13
```

```

10     input_images = torch.cat([input_images,
11                             data["pred_normals"][:, :model_cfg.data.input_images, ...]],
12                             dim=2)
13
14 # Get camera to center depth
15 if model_cfg.data.origin_distances:
16     input_images = torch.cat([input_images,
17                             data["origin_distances"][:, :model_cfg.data.input_images, ...]],
18                             dim=2)

```

Listing 6: Splatter Image input tensor preparation code

For the training code, for the base non-LoRA training script, two changes had to be made. The first change was to model loading, to add support for training from existing HuggingFace models from the base Splatter Image project.

CODEBLOCK

ADDED HF LAODING, PERFORMS GRAFT

The second change was similar to that of the evaluation script, updating input preparation, the resulting updated code block is identical to that in Listing 6.

For the LoRA-specific training script, 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 optimizer 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 dependencies into each VM, using requirements files stored alongside the scripts in `/geometry-priors`, and then execute the prior generation notebooks in order to generate 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 pregenerated dataset remains available. Models use the pregenerated datasets via a custom loader we implemented, taht slots in into all of the exisitng training/evlauation 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 EX-AMPLE 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 concrete 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 manny 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 WHTEHR YOU WNAT TO START THE MODEL WITH EXISITNG WEIGHTS (if so grafted) WEIGHTS FUSED AT SAVE

TRAINING IS RUN USING DEFAULT SPALTTER IAMGE APRAMSTERS, Namely: LIST OPTIM-SIER 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 7. 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.]

For novel view synthesis evaluation, PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index) and LPIPS (Learned Perceptual Image Patch Similarity) are standard metrics used to evaluate image quality from different perspectives[39]. PSNR is calculated using the mean squared error and is commonly used to quantify reconstruction quality, such that a high PSNR suggests higher reconstruction quality. Meanwhile, SSIM depends on 3 metrics mimicking the human visual perception system: luminance, contrast and structure, where an SSIM score close to 1 indicates high similarity while a score closer to -1 indicates low similarity. Lastly, unlike pixel-wise metrics like PSNR and SSIM that assume pixel independence, LPIPS measures the perceptual similarity between images by comparing their features extracted from a deep neural network, where a low LPIPS score indicates that the compared images are perceptually similar to humans.

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: rank, alpha and dropout (as mentioned in section 2.2.1).

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

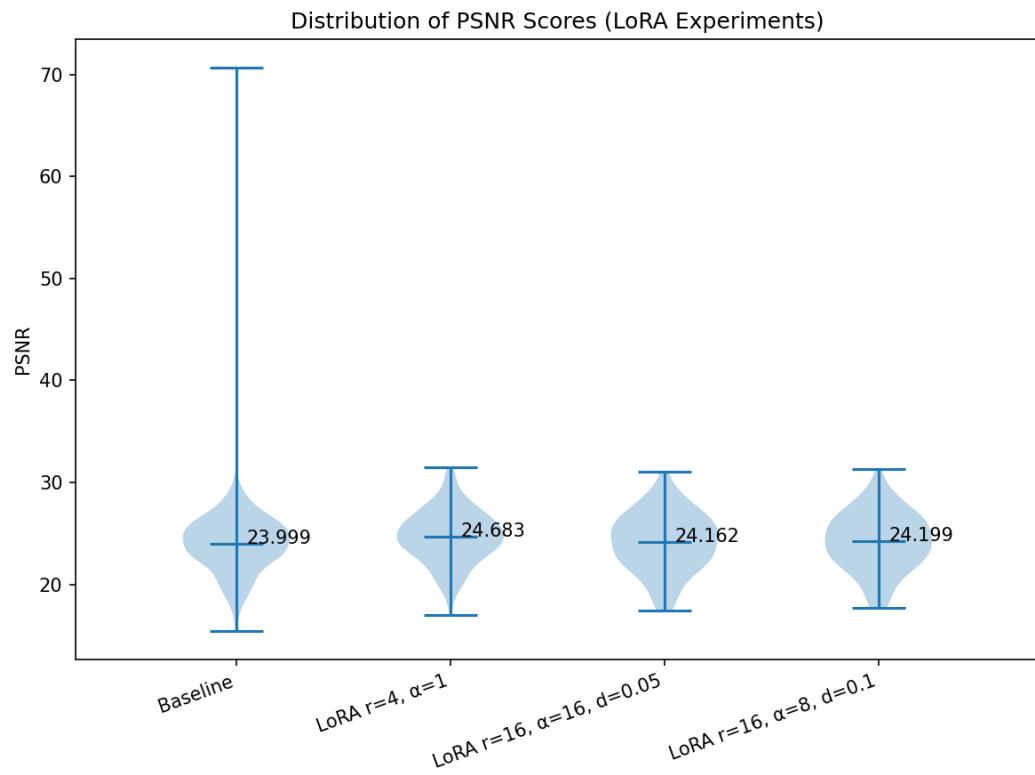


Figure 16: PSNR distribution for different LoRA configurations

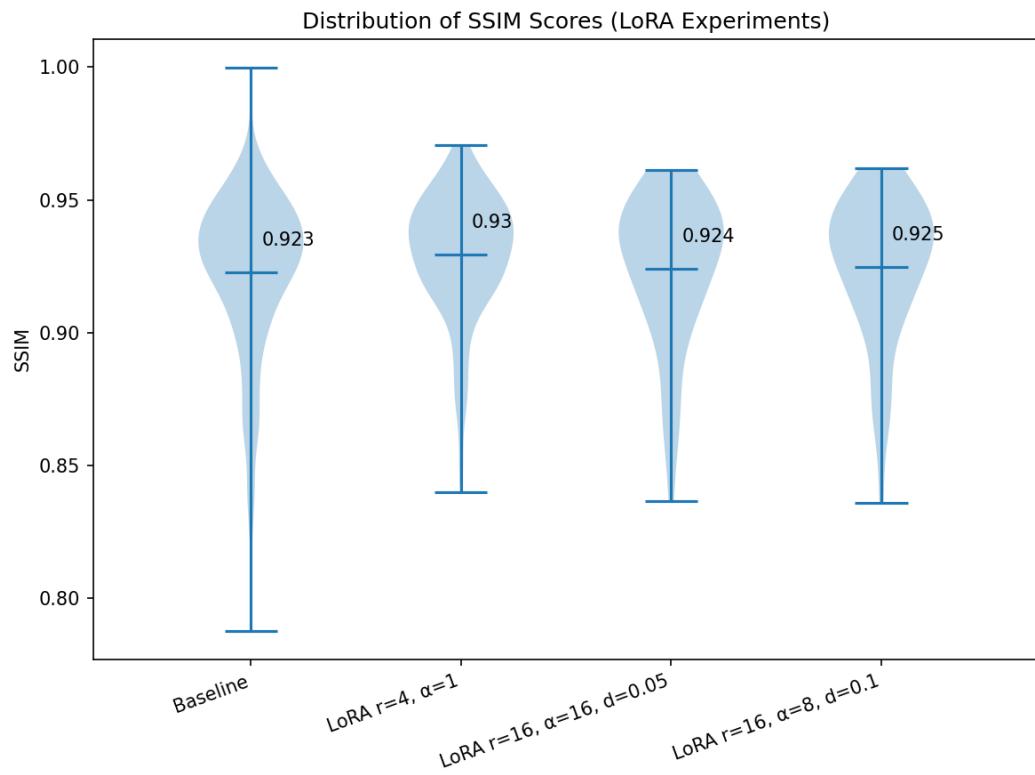


Figure 17: SSIM distribution for different LoRA configurations

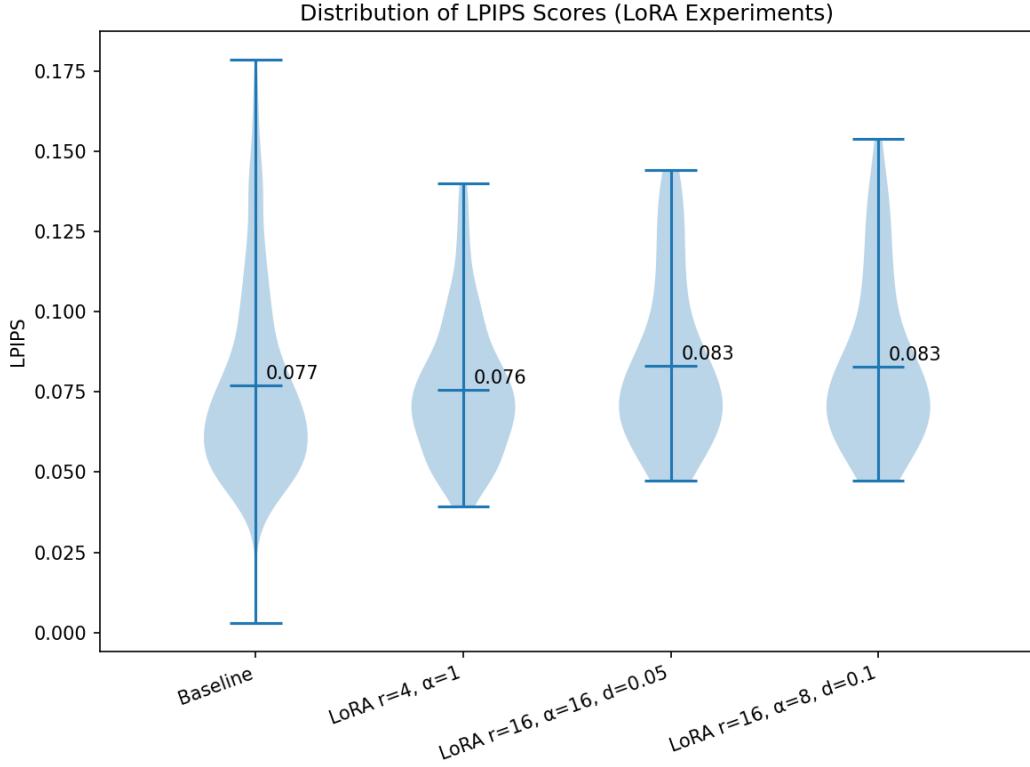


Figure 18: LPIPS distribution for different LoRA configurations

From figures 16, 17 and 18, we can see that all changed models now perform more consistently, as shown by the lower variance in all metrics as compared to the baseline. The PSNR metrics in figure 16 and the SSIM metrics in figure 17 also suggest better image reconstruction overall due to having higher mean PSNRs and SSIMs than the baseline, with the configuration $r = 4, \alpha = 1$ performing the best with the highest mean PSNR and SSIM. This configuration also has a lower mean LPIPS score than the baseline, further supporting our claim that this configuration is best for augmenting the model, while the other 2 configurations have higher mean LPIPS scores.

(significance testing?? idk anymore)

3.4.2 FiLM experiment results

To evaluate whether integrating different geometry priors into the model with FiLM layers (as described in TODO cite section) is effective, we performed a set of experiments testing different combinations of geometry priors. We measured the performance using PSNR, SSIM and LPIPS.

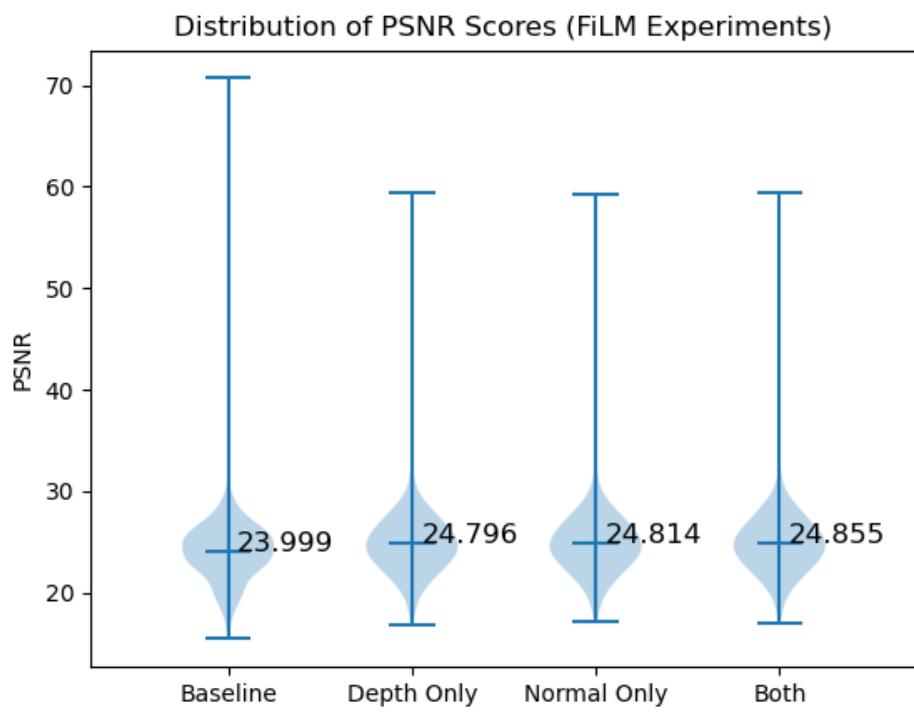


Figure 19: PSNR distribution for different combinations of added geometry priors with FiLM

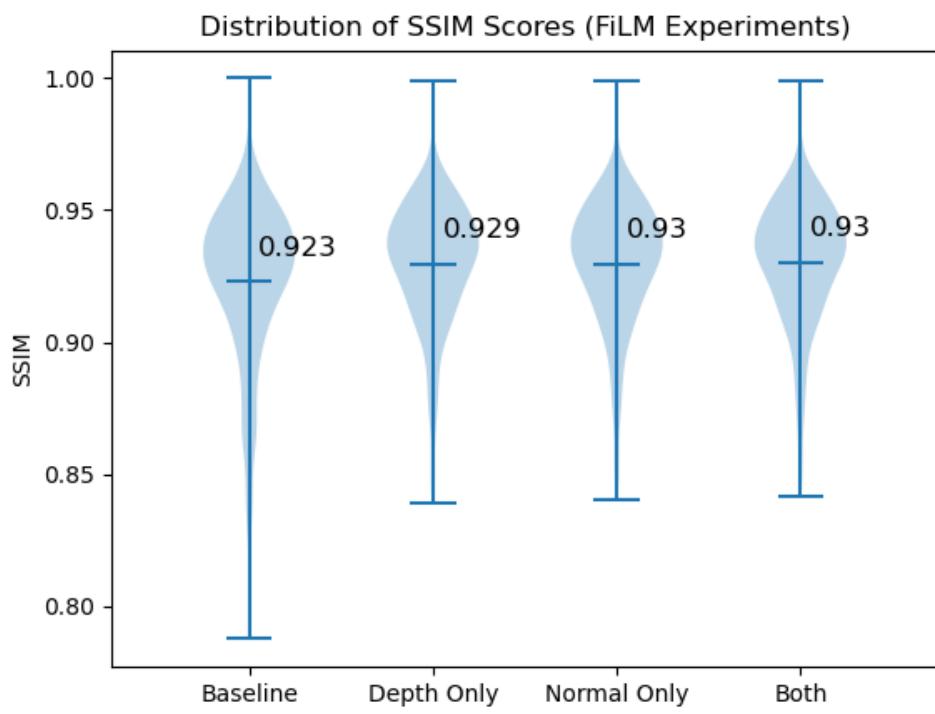


Figure 20: SSIM distribution for different combinations of added geometry priors with FiLM

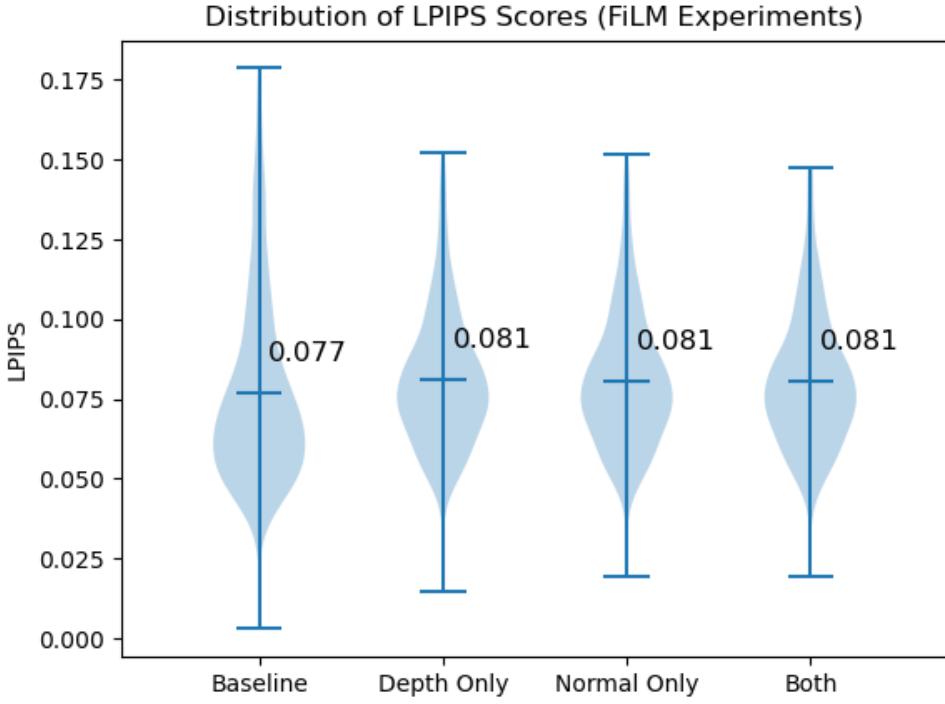


Figure 21: LPIPS distribution for different combinations of added geometry priors with FiLM

Similar to the LoRA experiment results in 3.4.1, using FiLM to augment the model with geometry priors makes the model perform more consistently, as shown by the lower variance in all metrics as compared to the baseline in figures 19, 20 and 21. All augmented models have a higher mean PSNR and SSIM from figures 19 and 20 respectively which suggests better image reconstruction quality, with the model adding both depth and normal priors performing the best. However, all augmented models have a higher LPIPS score than the baseline 21, suggesting that the difference between the ground truth and the images generated by the model perceived by humans are larger for augmented models than the baseline. We conclude that integrating these priors with FiLM layers may not be beneficial in some cases, for example cases where successful image reconstruction is defined as being similar to the ground truth based on a human’s perception.

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 (espoecially more priors aoriented to hidden areas, could bring up planes again) implementing and comapring cross-attention vs FiLM for multimodal data trying longer training times trying alternative datasets Trying alternatigve metrics for 3D reconstruction, like Chaumfeer distance, for exampel could sample off bunch of points of SRN cars meshes to genrate point cloud, then treat gaussians as point cloud and do comaprison

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?

Report Writing Contributions:

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: Alex

Section 3.2: Radhika

Section 3.3: FILL IN

Section 3.4: Radhika and Alex

Section 3.5: FILL IN

Section 4.1: FILL IN

Section 4.2: FILL IN

Section 4.3: FILL IN

Image collection: Kacper and Radhika and Alex

Citation collection: Radhika and Alex

Technical Contributions:

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 and Radhika

Splatter Image Eval Modification: Kacper

Optimised depth generation: Kacper

Optimised normals generation: Kacper and Alex

Optimised segmentation generation: Radhika

Splatter Image `cars_priors` custom Dataset: Kacper
HuggingFace dataset data pipeline: Kacper
Results processing code: Alex
Graphing code: Alex
Testing custom Dataset: Kacper
Testing model: Kacper
Testing LoRA configurations: Radhika

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, Zhenghe 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, Zhenghe 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.
- [39] Jiahui Zhang, Yuelei Li, Anpei Chen, Muyu Xu, Kunhao Liu, Jianyuan Wang, Xiao-Xiao Long, Hanxue Liang, Zexiang Xu, Hao Su, Christian Theobalt, Christian Rupprecht, Andrea Vedaldi, Kaichen Zhou, Paul Pu Liang, Shijian Lu, and Fangneng Zhan. Advances in feed-forward 3d reconstruction and view synthesis: A survey, 2025.