# Comprehensive Analysis of Modern 3D Reconstruction Techniques

Aditya Rathor Aditya Sahani Dishit Sharma Souvik Maji Veeraraju Elluru
Indian Institute of Technology Jodhpur
{b22ai044, b22cs003, b22cs082, b22cs089, b22cs080}@iitj.ac.in

## Abstract

*This technical report presents a comprehensive and systematic analysis of contemporary 3D reconstruction methodologies, examining their theoretical foundations, implementation challenges, and performance characteristics. We investigate a diverse spectrum of approaches including traditional geometry-based methods (Structure from Motion, Multi-View Stereo), volumetric techniques (Space Carving), and emerging neural representations (Neural Radiance Fields, Gaussian Splatting). The comparative analysis explores the fundamental trade-offs between reconstruction fidelity, computational efficiency, and scene representation flexibility across these paradigms. By contextualizing these techniques within a unified framework, we meticulously identify critical bottlenecks in current 3D reconstruction pipelines and outline promising research directions for addressing persistent challenges in complex scene reconstruction. Our thorough investigation reveals that while neural implicit representations offer unprecedented reconstruction quality for intricate scenes, hybrid strategies that combine both geometric constraints and learned priors demonstrate substantial robustness across diverse environmental conditions. The code is made available at the GitHub repository and the demo can be viewed at this HuggingFace website.*

## 1. Introduction

Three-dimensional reconstruction—the process of inferring 3D geometric structure from 2D observations—remains a fundamental yet challenging problem in computer vision with applications spanning autonomous navigation, cultural heritage preservation, medical imaging, and immersive media production. Despite remarkable progress in sensor technologies and computational power, generating accurate, complete, and photorealistic 3D models from unconstrained imagery continues to pose significant technical difficulties.

Over the past decade, the field has transitioned from classical multi-view geometry paradigms rooted in explicit feature matching and geometric constraints to powerful data-driven methods that employ deep neural networks to learn implicit representations of scenes. This evolution aligns with a broader trend toward replacing hand-crafted algorithms with learning-based architectures capable of directly extracting complex spatial relationships from data. The shift has been accelerated by the limitations of traditional approaches in handling challenging scenarios involving non-Lambertian surfaces, thin or repetitive structures, and varying illumination conditions.

In this report, we examine the theoretical underpinnings and practical implementations of a spectrum of state-of-the-art 3D reconstruction approaches: Structure from Motion (SfM), incremental SfM, Multi-View Stereo (MVS), Graph Attention Networks, Space Carving, Generative Adversarial Network (GAN)-based reconstruction, Neural Radiance Fields (NeRF), and Gaussian Splatting. By investigating these methods through the lens of their representational capacity, computational demands, and reconstruction quality across a range of real-world scenes, we aim to provide a nuanced perspective on the strengths and limitations of each paradigm.

Our motivation is driven by the understanding that no single approach universally dominates across all capture conditions and scene types. Each method has its own unique benefits and constraints that render it particularly suitable for specific reconstruction tasks. By methodically analyzing these techniques, we seek to illuminate the present state of 3D reconstruction research, highlighting the progress made and underscoring the potential avenues for future innovation.

## 2. Related Works

### 2.1. Structure from Motion and Multi-View Stereo

Structure from Motion (SfM) is one of the most long-standing approaches to reconstructing 3D geometry from unstructured image collections. Early breakthroughs by Snavely et al. [1] showcased the feasibility of large-scale reconstruction by harnessing community photo collections. Subsequent systems such as COLMAP [2] became ubiqui-

tous benchmarks for robust and scalable SfM. Incremental SfM methods, as seen in VisualSFM [3], progressively refine reconstructions by adding cameras and points to an initial estimate, thereby improving resilience to outliers at the expense of significant computational overhead.

Once the camera parameters and sparse point clouds are derived via SfM, Multi-View Stereo (MVS) methods are typically employed to generate dense reconstructions. Furukawa and Ponce's Patch-based Multi-View Stereo (PMVS) [4] was a seminal contribution, systematically expanding and filtering patch correspondences to build dense and accurate point clouds. More recent pipelines, including the MVS modules in COLMAP [5], integrate probabilistic depth map fusion, demonstrating improved robustness and clarity. Learning-based methods for MVS [6, 7] leverage neural architectures to extract more robust features and refine matching cost volumes, particularly valuable in challenging scenarios with low-textured regions.

### 2.2. Volumetric and Space Carving Approaches

Volumetric reconstruction leverages 3D grids, discretizing the scene into voxels to estimate surface geometry. Space Carving, introduced by Kutulakos and Seitz [8], frames reconstruction as identifying a photo-consistent subset of voxels that explain observed imagery. The original deterministic approach was extended to probabilistic [10] and octree-based formulations [11] to mitigate computational overhead and memory demands.

Volumetric fusion strategies, such as KinectFusion [12], further advanced real-time reconstruction by recursively updating a truncated signed distance function (TSDF) as new sensor data streams in. Although these methods excel with dense RGB-D inputs, they are generally less suited for purely image-based reconstruction, given the difficulty of estimating pixel-level depth accurately in scenes with complex geometry or reflective surfaces.

### 2.3. Learning-Based Approaches

Deep learning has ushered in novel paradigms for image-based 3D reconstruction, transcending the constraints of traditional geometry-based methods. Graph Attention Networks [13, 14] model scene information as graph-structured data, employing attention mechanisms to capture both local details and global spatial relationships. These techniques excel at mesh generation and offer strong generalization in complex environments.

Generative Adversarial Networks (GANs) [15, 16] have also been harnessed for reconstructing objects from minimal observations. By learning distributions of plausible 3D shapes, GAN-based solutions can produce complete reconstructions even when the input is sparse or partially occluded. However, their performance tends to degrade in novel scenes that deviate substantially from the training distribution, limiting their applicability in unconstrained real-world contexts.

### 2.4. Neural Implicit Representations

Neural implicit representations encode a scene's geometry and appearance directly in the weights of a neural network. Among these, Neural Radiance Fields (NeRF) [18] have sparked significant interest by synthesizing photo-realistic novel views from only a sparse set of input images. NeRF models learn a continuous volumetric function parameterized by an MLP, mapping 3D coordinates and viewing directions to color and density. Although they deliver unparalleled fidelity in terms of novel view synthesis, training can be time-intensive, and the approach is typically restricted to static scenes.

Subsequent work aimed to accelerate training and rendering (e.g., Instant NGP [20]) or handle dynamic scenes (e.g., D-NeRF [21]). Gaussian Splatting [22] combines implicit neural representations with explicit 3D primitives, representing scenes as learnable Gaussian primitives that achieve high-quality rendering with substantially enhanced efficiency. Furthermore, its explicit parameterization facilitates intuitive scene editing and manipulation.

### 2.5. Semantic-Aware Gaussian Splatting

Recent advancements have integrated semantic understanding into Gaussian splatting to produce semantically rich 3D reconstructions. For instance, **LangSplat** [23] combines language features from CLIP with learnable 3D Gaussian primitives, enabling open-vocabulary queries and real-time rendering. By leveraging a scene-specific language autoencoder and hierarchical semantic cues from models such as SAM, LangSplat achieves precise object segmentation and substantial speed improvements over prior methods. These techniques bridge the gap between accurate geometric modeling and high-level scene understanding, paving the way for interactive and semantically aware 3D applications.

## 3. Problem Definition

Recovering three-dimensional structure from a collection of images involves addressing fundamental ambiguities inherent in the 3D-to-2D projection process. Formally, let:

$$I = \{I_1, I_2, \ldots, I_n\}$$

represent a set of images acquired from unknown or partially known viewpoints. The objectives are to estimate:

1. The camera parameters

$$C = \{C_1, C_2, \ldots, C_n\},$$

comprising intrinsics (e.g., focal length) and extrinsics (e.g., rotation, translation).

2. A 3D representation of the scene

$$S,$$

which, when rendered from $C$, faithfully reproduces the input images $I$.

This reconstruction problem is inherently ill-posed: countless 3D configurations can yield similar 2D projections without additional constraints or priors. These priors may arise from geometric assumptions (e.g., multi-view consistency or surface smoothness), or they may be learned from data, as in modern deep learning-based frameworks.

Choosing an appropriate representation $S$ is crucial. Traditional schemes like point clouds, meshes, and voxel grids explicitly model surfaces, while implicit approaches rely on continuous functions (e.g., signed distance fields, occupancy networks). Neural radiance fields and other implicit neural architectures can model complex scene geometry and appearance with high fidelity, albeit often at significant computational expense. Hybrid strategies fuse advantages from multiple representations, seeking an optimal balance between accuracy, memory usage, and rendering speed.

Crucial challenges that persist include establishing reliable correspondences, handling occlusions, managing non-Lambertian reflectance, coping with irregular or sparse viewpoint distributions, and scaling to large, complex environments. As the breadth of potential applications for 3D reconstruction expands—ranging from autonomous vehicle perception to cinematic visual effects—the quest for faster, more robust, and more flexible reconstruction algorithms remains a central research goal.

# 4. Methodologies

## 4.1. Structure from Motion

**Methodology & Mathematical Foundation**

The Structure from Motion (SfM) pipeline relies on a series of mathematical and algorithmic steps. It begins with the use of the pinhole camera model, where 3D points $\mathbf{X} \in \mathbb{R}^3$ are projected onto 2D image points $\mathbf{x} \in \mathbb{R}^2$ using the equation $\mathbf{x} = K[R|t]\mathbf{X}$. Here, $K$ is the intrinsic matrix of the camera, and $R$, $t$ define the rotation and translation matrices that together form the projection matrix $[R|t]$ representing the camera pose.

Next, features are extracted and matched across image pairs. Keypoints are detected and described, typically using SIFT, and then matched using FLANN, with Lowe's ratio test applied to filter ambiguous matches. Using these matches, the Fundamental Matrix $F$ is estimated through RANSAC, and from it, the Essential Matrix is computed via $E = K^\top FK$. This Essential Matrix is then decomposed using Singular Value Decomposition (SVD) to extract the relative camera pose $[R|t]$, and the correct configuration is selected using the cheirality condition, which ensures that reconstructed points lie in front of both cameras.

With the relative pose established, triangulation is performed to reconstruct 3D scene points. Given matched keypoints $\mathbf{x}_1$, $\mathbf{x}_2$ and their respective projection matrices $P_1$, $P_2$, the 3D point $\mathbf{X}$ is computed by solving a linear least squares problem that minimizes the reprojection error: $\min_{\mathbf{X}} \|\mathbf{x}_1 - P_1\mathbf{X}\|^2 + \|\mathbf{x}_2 - P_2\mathbf{X}\|^2$, typically using the DLT algorithm.

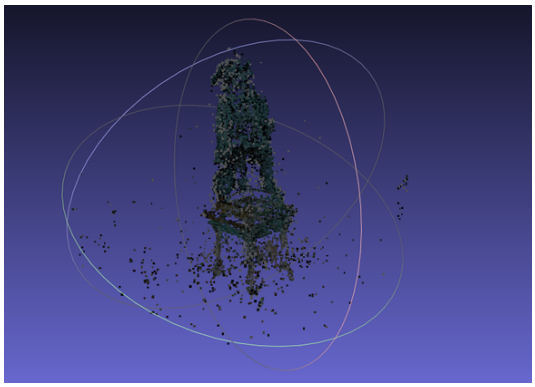
For integrating new views, the Perspective-n-Point (PnP) algorithm is used. Given known 3D-2D correspondences $\{\mathbf{X}_i, \mathbf{x}_i\}$, the goal is to estimate the camera pose such that $\mathbf{x}_i = K[R|t]\mathbf{X}_i$. This is solved using the `solvePnPRansac` algorithm, which provides robustness to outliers.

To refine both the 3D point positions and camera poses, bundle adjustment is employed. This is a nonlinear optimization process that minimizes the total reprojection error over all images and points, formulated as: $\min_{R_i,t_i,\mathbf{X}_j} \sum_{i,j} \|\mathbf{x}_{ij} - \pi(K[R_i|t_i]\mathbf{X}_j)\|^2$, where $\pi$ denotes the projection operator. This optimization is typically carried out using the Levenberg-Marquardt algorithm, implemented through `scipy.optimize.least_squares`.

The implementation is modular and includes several components. The image loader reads all images from a specified directory, loads the camera intrinsics from a file (e.g., `K.txt`), and optionally downsamples the images for efficiency. The feature matching module detects and matches keypoints using SIFT and FLANN. Motion estimation calculates the Essential Matrix and recovers the relative pose using `recoverPose`. Triangulation then computes 3D point locations from matched keypoints and projection matrices. For new frames, PnP is used to estimate the pose using 2D-3D correspondences. Reprojection error is computed to evaluate consistency, which also serves as the objective function for bundle adjustment. An optional bundle adjustment module jointly optimizes the poses and 3D points. Finally, the point cloud is filtered to remove outliers based on distance from the centroid and exported in the .ply format for visualization.

The output of the pipeline is a colored 3D point cloud saved as `res/<folder_name>.ply`, which can be visualized using tools like MeshLab or ParaView. Overall, the SfM system provides a robust method for recovering 3D structure from multiple calibrated 2D views. Its modular design enables easy extension for advanced tasks such as dense reconstruction, SLAM integration, or neural rendering.

(a) Real Image View



(b) Reconstructed SfM 3D View

Figure 1. Comparison between the original real image and the reconstructed 3D view from the Structure from Motion pipeline.

#### 4.1.1 Incremental Structure from Motion

Incremental Structure from Motion (SfM) is a sequential 3D reconstruction approach where the scene structure and camera parameters are recovered progressively, starting from an initial image pair. As new views are added, camera poses are estimated using existing 3D points and 2D feature correspondences, and new 3D points are triangulated. This bootstrap method enables robust and scalable 3D reconstruction, even in the presence of noise or partial occlusion.

The algorithm proceeds in the following stages:

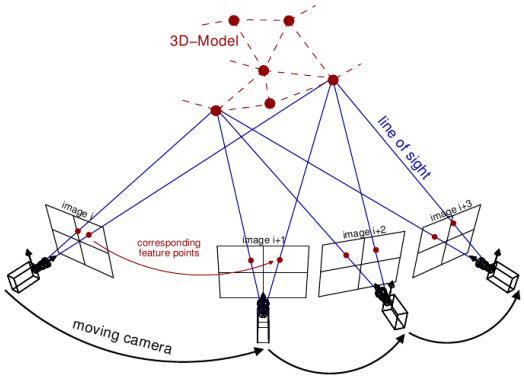



Figure 2. Pipeline of Incremental Structure from Motion.

1. **Initialization:** Start with two calibrated views and compute the initial sparse point cloud using triangulation.

2. **Pose Estimation:** Add new camera views by estimating their extrinsics using the Perspective-n-Point (PnP) algorithm on 2D-3D correspondences.

3. **Triangulation of New Points:** Once the new pose is estimated, new correspondences are triangulated using the newly added camera and previously registered ones.

4. **Expansion:** The model is progressively expanded by repeating the process for subsequent views.

**Mathematical Foundation**

Each camera is modeled using a projection matrix:

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{1}$$

where $\mathbf{K}$ is the intrinsic calibration matrix, and $\mathbf{R}, \mathbf{t}$ are the rotation and translation components (extrinsics).

Given 2D image correspondences $\mathbf{x}_1$ and $\mathbf{x}_2$ from two camera views with known projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$, the 3D point $\mathbf{X}$ is computed via triangulation by solving:

$$\mathbf{x}_1 = \mathbf{P}_1\mathbf{X}, \quad \mathbf{x}_2 = \mathbf{P}_2\mathbf{X} \tag{2}$$

For solving camera poses in later frames, we use the PnP formulation:

$$\text{minimize} \sum_i \|\mathbf{x}_i - \pi(\mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}_i)\|^2 \tag{3}$$

where $\pi(\cdot)$ denotes projection from 3D to 2D.

**Implementation Overview**

Our implementation is written in Python and uses OpenCV for PnP estimation and NumPy for matrix operations. The point cloud is visualized at intermediate stages using custom 3D plotting utilities. The key components of the code include:

- `findMatches`: Detects correspondences between feature sets across different images.
- `getCamera`: Solves for the camera pose using OpenCV's `solvePnP`, and constructs the projection matrix.
- `add_points3d` and `triangulate_and_add`: Perform triangulation for new points using existing and new views.
- `sfm`: Main driver function orchestrating the pipeline — loading data, managing the reconstruction loop, and visualizing output.

## 4.2. Space Carving

*Space Carving*, also known as *Voxel Carving* offers a powerful and intuitive volumetric approach that leverages silhouette information from multiple calibrated views. It operates under the principle of photo-consistency, iteratively removing regions of space that are inconsistent with the set of observed image silhouettes. Starting with an initial volumetric representation (typically a uniform voxel grid), the algorithm projects each voxel into all input views using known camera poses. Voxels that fall outside the object's silhouette in any view are eliminated, effectively "carving" away the empty space and approximating the object's visual hull. Compared to traditional multiview stereo methods, Space Carving requires no texture information and is particularly effective when silhouettes are well-defined and background separation is feasible. Although inherently limited by voxel resolution and sensitive to silhouette quality, it remains a widely used method for coarse-to-moderate resolution reconstructions in controlled environments, e.g., archaeological artifact modeling, robotics and navigation (shape understanding), and medical imaging (organ modeling from slices).

### 4.2.1 Dataset Preparation

The Space Carving reconstruction pipeline was evaluated using the Dino dataset [9], which consists of calibrated multi-view images and corresponding camera projection matrices. The following steps were performed in the data preparation phase:

- **Camera Matrix Loading**: The calibrated camera projection matrices $\{P_i\}$ were loaded from a provided MATLAB file (dino_Ps.mat). These matrices are essential for projecting 3D voxel coordinates into 2D image space.
- **Image Acquisition and Preprocessing**: All views were then loaded using OpenCV. The images are converted to RGB color space and normalized to the range [0,1]. These normalized images serve as the input for silhouette extraction.
- **Silhouette Extraction**: Silhouette masks were derived from the images by assuming a blueish background with RGB values approximately [0.0, 0.0, 0.75]. Foreground pixels are detected by thresholding the Euclidean distance in RGB space, with a threshold of 1.1. Binary masks are then refined using morphological opening with a $5 \times 5$ kernel to reduce noise and ensure smooth boundaries.

### 4.2.2 Algorithm and Architecture

Let's now go over the details of the core components of the Space Carving pipeline, from voxel grid creation to final point cloud extraction. A volumetric grid of resolution $120^3$ is initialized in normalized coordinates. Each voxel is stored in homogeneous coordinates $[\mathbf{x}, \mathbf{y}, \mathbf{z}, 1]^T$ to facilitate projection via matrix multiplication with the camera matrices. The grid is centered and scaled to encompass the approximate bounding volume of the target object. Now, for each voxel in the grid, we perform the following steps:

- Project into all image planes using the corresponding camera projection matrix $P_i$. This yields a 2D coordinate $(u, v)$, and the voxel is retained if it falls within the foreground region (i.e., silhouette) in all views.
- Then, the **occupancy score** is computed as the number of views in which the projected point lies within the silhouette. This score serves as a visibility metric and forms the basis for inclusion in the final 3D shape.
- Voxels with occupancy scores above the threshold $\tau$ are considered as part of the object's surface and extracted into a 3D point cloud, which is exported as a .csv file (with 3D coordinates) and a .vtr file for VTK-based visualization.

Mathematically, the projection and inclusion test are expressed as follows:

$$\mathbf{u}_{\text{hom}} = P_i \cdot \mathbf{X}, \tag{4}$$

$$\mathbf{u} = \left[ \frac{u_{\text{hom}}}{w_{\text{hom}}}, \frac{v_{\text{hom}}}{w_{\text{hom}}} \right] \tag{5}$$

$$\text{occupancy}(v) = \sum_{i=1}^{N} \mathbf{1}[P_i \cdot v \in \text{Silhouette}_i] > \tau \tag{6}$$

where $\tau$ is a threshold parameter (e.g., 4 out of $N$ views).

### 4.3. Pix2Vox

Pix2Vox is a learning-based approach for single-view and multi-view 3D object reconstruction. It maps 2D RGB images to 3D voxel grids and employs a context-aware fusion module to effectively integrate information from multiple views without relying on explicit pose estimation. Additionally, a **discriminator** was incorporated during training to subtly guide the model towards producing more realistic and detailed reconstructions by encouraging sharper object boundaries and finer structural details.

#### 4.3.1 Data Preparation

The ShapeNet [27] dataset, specifically the ShapeNetRendering and ShapeNetVox32 subsets provided by Stanford's CVGL group wass used. The rendering dataset contains RGB images of resolution 137×137×3 captured from multiple viewpoints around 3D CAD models, while the voxelized dataset provides corresponding ground-truth 3D volumes at a resolution of 32×32×32. During training, a series of data augmentations are applied to the input images to improve generalization. These include random cropping, background color randomization, color jittering (brightness, contrast, saturation), Gaussian noise addition, normalization, random horizontal flipping.

#### 4.3.2 Architecture

The **Encoder** is initialized with a VGG16 [31] model pretrained on ImageNet [30]. For each input image of size $224 \times 224 \times 3$, it extracts a $28 \times 28 \times 512$ feature tensor. These features are then processed through three additional `nn.Conv2D` layers, each followed by an `nn.ELU` activation function, to further embed semantic information relevant to 3D reconstruction.

The **Decoder** takes the 2D feature maps from the encoder and transforms them into coarse 3D volumes. It comprises four transposed convolutional layers (implemented throughout using `nn.ConvTranspose2d`). Each of the first three layers is followed by a ReLU activation function (`nn.ReLU` throughout), while the last layer is followed by a sigmoid activation function to constrain the voxel output to the range $[0, 1]$. The output is a voxel grid of dimensions $32 \times 32 \times 32$. The encoder-decoder pipeline that generates coarse 3D volumes from multiple input images in parallel. This design *eliminates the dependency on the order* of the input images and accelerates the reconstruction process.

Next, a **Context-Aware Fusion Module** selects high-quality reconstructions from the set of coarse 3D volumes (one per view) and generates a fused 3D volume. This mechanism effectively integrates information from all input views while avoiding long-term memory loss. This is done using a **Context Scoring Network** that takes the last two layers of the decoder as input and generates a score $m_r$ for the context $c_r$ of the $r$-th coarse volume, where $r$ denotes the view index. The context scoring network consists of five 3D convolutional layers, each with kernel size 3 and padding 1, followed by batch normalization and LeakyReLU activations. The number of output channels for the five layers are 9, 16, 8, 4, and 1, respectively.

The learned score $m_r$ is normalized across all views using the softmax function. The score at voxel location $(i, j, k)$ for view $r$ is computed as:

$$s_r^{(i,j,k)} = \frac{\exp\left(m_r^{(i,j,k)}\right)}{\sum_{p=1}^{n} \exp\left(m_p^{(i,j,k)}\right)}$$

These scores serve as weights for each view, and a weighted summation is performed over the coarse volumes to obtain the fused volume:

$$\mathbf{v}_f = \sum_{r=1}^{n} s_r \, \mathbf{v}_r^{(c)}$$

Lastly, the **Refiner** corrects errors in the fused 3D volume and enhances the overall quality of the reconstruction. It follows a U-Net-like architecture [33] that preserves local structure through skip connections between the encoder and decoder. The encoder part of the refiner consists of three 3D Convolutional layers, each with a $4 \times 3$ kernel and padding of 2. Each layer is followed by batch normalization, a LeakyReLU activation, and a Max pooling layer with kernel size 2. The convolutional layers produce 32, 64, and 128 output channels, respectively. Two fully connected layers with output dimensions of 2048 and 8192 follow the encoder. The decoder of the refiner contains three transposed 3D convolutional layers, each with a $4 \times 3$ kernel, padding of 2, and stride 1. These layers reconstruct the refined 3D volume using the high-level features extracted by the encoder.

In the efforts to enhance the reconstruction resolution, a GAN [34] based framework is employed by incorporat-

| Component | Details |
|---|---|
| Encoder | Input: $224 \times 224 \times 3$ RGB image<br>VGG16 pretrained on ImageNet<br>3 Conv2D layers with ELU activations<br>Output: $28 \times 28 \times 512$ feature tensor |
| Decoder | Input: 2D feature maps<br>4 transposed Conv3D layers<br>ReLU in first 3 layers, Sigmoid in last<br>Output: $32 \times 32 \times 32$ voxel grid |
| Merger (Fusion) | Input: coarse volumes from multiple views<br>Context scoring network with 5 Conv3D layers<br>Kernel size: $3^3$, Padding: 1<br>LeakyReLU + BatchNorm in each layer<br>Scores normalized via softmax across views<br>Weighted sum produces fused volume |
| Refiner | Input: fused 3D volume<br>Encoder: 3 Conv3D layers with LeakyReLU + MaxPool<br>Followed by two FC layers (2048, 8192 units)<br>Decoder: 3 transposed Conv3D layers<br>U-Net style skip connections used |
| Discriminator | Input: $32 \times 32 \times 32$ voxel volume<br>4 Conv3D layers (kernel $4^3$, stride 2, padding 1)<br>Channels: $32 \rightarrow 64 \rightarrow 128 \rightarrow 256$<br>BatchNorm3D + LeakyReLU activations<br>Final FC layer with Sigmoid activation |

Table 1. Complete summary of Pix2Vox architecture (with **discriminator** and **refiner** components.

ing a **discriminator** into the training pipeline. It receives a single-channel input volume of size $32 \times 32 \times 32$ and processes it through four 3D convolutional layers with kernel size 4, stride 2, and padding 1. Each layer is followed by (a 3D) BatchNorm layer followed by LeakyReLU activations. These layers progressively increase the channel depth from 32 to 256 and reduce the spatial size to $2 \times 2 \times 2$. The resulting feature map is flattened and passed through a fully connected layer with a sigmoid activation, producing a scalar value that indicates the realism of the input. The refiner (acting as the generator, here) is trained to minimize voxel-wise reconstruction loss while also attempting to fool the discriminator, encouraging it to produce outputs that resemble real 3D shapes. A summary of the complete architecture for Pix2Vox is presented in Table 1.

#### 4.3.3 Loss Function

The loss function for the refiner uses the reconstructed and original voxel. In the below equation, N is the number of voxels in the ground truth images. For ex: for 32*32*32 voxel sizes, N is 32768.

$$\mathcal{L}_{\text{refiner}} = \frac{1}{N} \sum_{i=1}^{N} [\text{gt}_i \log(p_i) + (1 - \text{gt}_i) \log(1 - p_i)]$$

Next, to improve structural realism, the refined volume is passed through a discriminator, and an **adversarial loss** is computed to encourage outputs that are indistinguishable from real samples, similar to how it is done in a typical GAN-style training:

$$\mathcal{L}_{\text{gen}} = \text{BCE}(D(V_{\text{refined}}), 1)$$

The final loss used to train the generator (refiner) combines the standard voxel-level BCE loss and the adversarial component with regularizer coefficients.

$$\mathcal{L}_{\text{total}} = \lambda_1 \cdot \mathcal{L}_{\text{refiner}} + \lambda_2 \cdot \mathcal{L}_{\text{gen}}$$

The discriminator itself is trained to distinguish between real ground truth volumes (V_gt) and generated ones (V_refined) bringing the adversary between the refiner(generator) and discriminator.

$$\mathcal{L}_{\text{disc}} = \text{BCE}(D(V_{\text{gt}}), 1) + \text{BCE}(D(V_{\text{refined}}), 0)$$

#### 4.3.4 Training

During the training phase, a batch size of 64 was used alongside reshaping the input images to a resolution of $224 \times 224$. The output was a voxelized 3D reconstruction of size $32 \times 32 \times 32$. Further, we used the Adam optimizer with momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The initial learning rate was set to 0.01 which decayed after 50 epochs. The main training process comprises of two phases:

- **Phase 1:** All the networks except the context-aware fusion module was trained using the single-view images, for 150 epochs (which took around 10 hours).
- **Phase 2:** In the next phase, we jointly trained the entire network including the context-aware fusion module, using a random number of input views, for 100 epochs (which took another 8 hours).

**Evaluation Metric**: IOU provides a voxel-wise evaluation, measuring how well the predicted shape aligns with the real one.Has value between 0 to 1. Higher IoU values indicate better reconstruction results.

$$\text{IoU} = \frac{\sum_{i,j,k} \mathbb{I}(p_{i,j,k} > t) \cdot \mathbb{I}(\text{gt}_{i,j,k})}{\sum_{i,j,k} \mathbb{I}\left(\mathbb{I}(p_{i,j,k} > t) + \mathbb{I}(\text{gt}_{i,j,k})\right)}$$

### 4.4. Neural Radiance Fields

Let's move on to learning based models which now use radiance fields and incorporate classical rendering techniques. The goal of neural radiance fields is to be able to learn a continuous scene representation from sparse set of views. This enables us to predict a large number of novel

views of a particular scene from a limited number of input views.

The ray rendering algorithm forms the crux of any good NeRF model. We elucidate our implementation of a simpler version of the full algorithm in the original paper [18]. The algorithm's main motive is to be able to replicate real light falling on the scene from different views. It takes in the current NeRF model along with a set of camera rays defined by their origins and directions. It samples a fixed number of points along each ray between specified near and far bounds. We use stratified sampling as done in [19], where each ray segment is randomly perturbed within its bin. The resulting depth samples are used to compute 3D query points along the ray. These points are passed through the NeRF model, which returns color ($\mathbf{c}(\mathbf{r}(t), \mathbf{d})$) and density values ($\sigma(\mathbf{x})$). This volume density can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location $\mathbf{x}$. To compute the final pixel color, classical volume rendering [17] is applied: densities are converted to alpha values using the optical model, and transmittance weights are computed cumulatively along the ray. The expected color $\mathcal{C}(\mathbf{r})$ of a camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds $t_n$ and $t_f$ is given by:

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

where the accumulated transmittance $T(t)$ is defined as:

$$T(t) = \exp\left( - \int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds \right)$$

Hence, the color contributions from all samples are accumulated using their respective weights. To handle synthetic backgrounds (e.g., white backgrounds), a regularization term based on the total opacity is added. The complete algorithm is summarized in Algorithm 1.

**Algorithm 1** Ray Rendering Algorithm in NeRF (Modified)

**function** RENDERRAYS(model, ray_origins, ray_dirs, $h_n$, $h_f$, num_bins)
  **Input:** Neural field model, ray origins $\mathbf{o}$, directions $\mathbf{d}$, near/far bounds $h_n, h_f$, number of bins $N$
  $t \leftarrow \text{linspace}(h_n, h_f, N)$
  Compute midpoints between adjacent $t_i$'s
  $t_i \leftarrow$ stratified sample in $[t_i, t_{i+1}]$
  **Compute bin sizes: (summation instead of integration)**
  $\Delta_i \leftarrow t_{i+1} - t_i$ for $i = 1, \ldots, N-1$
  $\Delta_N \leftarrow \infty$
  **Sample points along ray:**
  $\mathbf{x}_i = \mathbf{o} + t_i \cdot \mathbf{d}$
  **Reshape for batch evaluation:**
  Flatten $\mathbf{x}_i$ and $\mathbf{d}$ into 2D tensors
  **Query model:**
  $(\mathbf{c}_i, \sigma_i) \leftarrow \text{model}(\mathbf{x}_i, \mathbf{d})$
  **Compute volume rendering weights:**
  $\alpha_i = 1 - \exp(-\sigma_i \cdot \Delta_i)$
  $T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$ ▷ Cumulative transmittance
  $w_i = T_i \cdot \alpha_i$
  **Aggregate final color:**
  $\mathbf{C} = \sum_{i=1}^{N} w_i \cdot \mathbf{c}_i$
  **Regularize with white background:**
  $\mathbf{C} \leftarrow \mathbf{C} + (1 - \sum_i w_i)$
  **return** $\mathbf{C}$
**end function**

#### 4.4.1 Data Preparation

The Lego crane dataset was used from these publicly available pickle files - (**cite**). The training dataset contains 9-dimensional vectors where the ray directions, ray origins and ground-truth pixel values take up 3 dimensions each. The testing dataset is similar, except we only have ray directions and ray origins, for which the pixel values need to be predicted. The testing dataset is much larger than the training dataset as the main goal is come up with a neural network that can synthesize a large number of *novel* views for the scene it has learned. Without any data-preprocessing steps, we prepare the DataLoader with batch sizes of 1024 rays as done in [18].

#### 4.4.2 Architecture

The architecture implemented is a modified version of the baseline architecture provided in [19]. Define

$$\mathbf{x} \in \mathbb{R}^3, \quad \mathbf{d} \in \mathbb{R}^3, \quad \gamma(.)$$

as the spatial coordinate, viewing direction, and positional encodings respectively. The architecture mainly involves two particular branches - $\gamma_1(\mathbf{x})$ and $\gamma_2(\mathbf{d})$, for the positional encodings to yield position-based and direction-based features respectively. Both $\gamma_1$ and $\gamma_2$ are simple MLPs, consisting of `nn.Linear` Layers and `nn.ReLU` activations. The architecture is summarized in Table 2 and Figure 3.

#### 4.4.3 Training Procedure

The training and inference strategy for Neural Radiance Fields is quite distinct from typical deep learning models. Since these fields (and hence the model) are fixed for a particular 3D scene, given sparse input views and known camera positions, the training objective is to *overfit* the model for this scene. The same weights can not be borrowed over

| Component | Details |
|---|---|
| Input Position | $6 \times \gamma_p + 3$ |
| Input Direction | $6 \times \gamma_d + 3$ |
| Position MLP ($F_{pos}$) | 8 layers, Linear(384) + ReLU<br>last: Linear $\rightarrow 3D + 1$ |
| Direction MLP ($F_{dir}$) | 5 layers, Linear(128) + ReLU<br>last: Linear $\rightarrow D$ |

Table 2. Summary of the NeRF model architecture. $D$ is the number of color basis outputs, and $\gamma_p$, $\gamma_d$ are positional encoding dimensions.

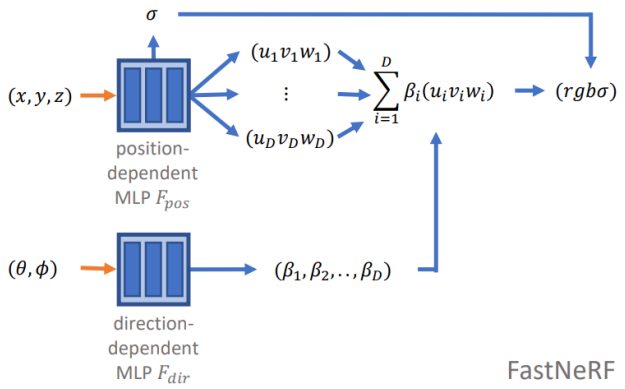



Figure 3. Visual Representation of the forward pass for our NeRF implementation. The architecture used is a modification of this architecture proposed in [19] (in terms of the MLPs used, the summary of which is already provided in 2). The ray rendering algorithm elucidated above is used to predict the deep radiance map $(u, v, w)$.

for a different scene. This unique procedure enables us to obtain high fidelity, novel views during the final rendering process at inference time based on this neural network that entirely represents that single scene.

The input contains ray origins and ray directions obtained from the data preparation steps. The synthesizing of views is performed by querying 5D coordinates along camera rays (starting from corresponding ray origins and along ray directions) and by using classical volume rendering techniques to project the output colors and densities into an image. The entire goal of training the radiance fields is to be able to obtain a continuous distribution for these colors and image densities. The loss functions that was minimized is a simple $\mathbb{L}2$ loss between the reconstructed ($\mathbf{c}_i$) and ground truth ($\hat{\mathbf{c}}_i$) pixel values, given by the following equation

$$\mathcal{L} = \sum_i \|\hat{\mathbf{c}}_i - \mathbf{c}_i\|_2^2$$

**Hyperparameters**: In our experiments, we used an `Adam` optimizer with initial learning rate as 5e-4, and a `MultiStepLR` learning rate scheduler with decay milestones as (2, 4, 8), and a gamma of 0.5. The model was trained for approximately 20-25 epochs (∼5 hours). All the other hyperparameters were used with their default values. Exhaustive hyperparameter tuning is left to the readers.

**Inference**: Each ray is sampled between near plane $h_n$ and far plane $h_f$ (computed based on dataset) using N=192 bins for the summation along the ray. This takes about 10 seconds. Note that both training and testing were performed on a single NVIDIA A6000 GPU.

## 4.5. Semantic Gaussian Splatting

**Methodology.** In this section, we detail the methodology employed to reimplement LangSplat, a novel approach for constructing a 3D language field that supports precise and efficient open-vocabulary querying in 3D scenes, as proposed by Qin et al. (2024). Unlike prior methods that rely on Neural Radiance Fields (NeRFs), LangSplat leverages 3D Gaussian Splatting for efficient rendering and integrates hierarchical semantics using the Segment Anything Model (SAM) to enhance language feature precision. Our reimplementation follows the framework depicted in Figure 2 of the original paper, addressing key challenges in 3D language field modeling, such as rendering efficiency, memory constraints, and point ambiguity. The methodology is structured into four main components: hierarchical semantics extraction with SAM, scene-specific language autoencoder training, 3D language Gaussian Splatting, and open-vocabulary querying.

### 4.5.1 Hierarchical Semantics with SAM

To achieve pixel-aligned language embeddings with clear object boundaries, we utilize the Segment Anything Model (SAM) to extract hierarchical semantics from 2D training images. SAM provides segmentation maps at three distinct semantic levels—subpart, part, and whole—addressing the point ambiguity issue where a single 3D point may belong to multiple semantic concepts (e.g., a point on a cat's ear relates to "ear," "head," and "cat").

For each input image $\boldsymbol{I}_t \in \mathbb{R}^{3\times H\times W}$, where $H$ and $W$ denote the height and width, we input a regular grid of $32 \times 32$ point prompts into SAM. This generates three sets of masks: $\boldsymbol{M}_t^s$, $\boldsymbol{M}_t^p$, and $\boldsymbol{M}_t^w$, corresponding to subpart, part, and whole levels, respectively. Redundant masks are filtered based on predicted IoU scores, stability scores, and overlap rates, ensuring distinct and comprehensive segmentation maps at each level.

For each pixel $v$ in image $\boldsymbol{I}_t$, we identify the mask $\boldsymbol{M}^l(v)$ it belongs to at semantic level $l \in \{s, p, w\}$ and compute its language embedding using the CLIP image encoder $V$:

$$\boldsymbol{L}_t^l(v) = V\big(\boldsymbol{I}_t \odot \boldsymbol{M}^l(v)\big), \tag{7}$$

where $\odot$ represents element-wise multiplication (masking the image to the region defined by the mask). This results in a set of pixel-aligned language embeddings $\boldsymbol{L}_t^l \in \mathbb{R}^{D\times H\times W}$, with $D = 512$ being the CLIP feature dimension. Each pixel thus has three embeddings, one per semantic level, enabling precise semantic representation without the need for additional regularization (e.g., DINO features) used in prior methods like LERF.

This SAM-based approach offers two advantages: (1) precise object boundaries improve the accuracy of the 3D language field, and (2) predefined semantic scales eliminate the need for multi-scale rendering during querying, enhancing efficiency.

#### 4.5.2 Scene-wise Language Autoencoder

Directly associating high-dimensional CLIP embeddings $\mathbb{R}^{512}$ with each 3D Gaussian is memory-intensive, especially given the millions of Gaussians required for complex scenes. To address this, we train a scene-specific autoencoder to compress these embeddings into a lower-dimensional latent space, reducing memory demands while preserving semantic information.

We collect all CLIP embeddings $\{\boldsymbol{L}_t^l(v)\}$ across all pixels $v$, images $t = 1,\dots,T$, and semantic levels $l \in \{s,p,w\}$. The autoencoder consists of an encoder $E : \mathbb{R}^D \to \mathbb{R}^d$ and a decoder $\Psi : \mathbb{R}^d \to \mathbb{R}^D$, where $d \ll D$. In our implementation, we set $d = 3$ as per the paper's findings for optimal efficiency and accuracy. The autoencoder is trained to minimize the reconstruction loss:

$$\mathcal{L}_{\text{ac}} = \sum_{l\in\{s,p,w\}} \sum_{t=1}^{T} \sum_{v} d_{\text{ac}}\big(\Psi\big(E\big(\boldsymbol{L}_t^l(v)\big)\big), \boldsymbol{L}_t^l(v)\big), \quad (8)$$

where $d_{\text{ac}}$ is a composite distance function combining L1 loss and cosine distance to ensure both magnitude and directional fidelity in reconstruction. After training, the compressed latent features are computed as:

$$\boldsymbol{H}_t^l(v) = E\big(\boldsymbol{L}_t^l(v)\big) \in \mathbb{R}^d, \quad (9)$$

serving as supervision targets for the 3D language Gaussians. This compression leverages scene-specific priors, as the variability of language features within a single scene is significantly lower than across CLIP's training dataset, enabling effective dimensionality reduction.

#### 4.5.3 3D Language Gaussian Splatting

We represent the 3D scene using a collection of 3D Gaussians, an explicit modeling approach that offers faster rendering compared to NeRF's implicit representation. Each Gaussian $G(x)$ is defined by a mean $\mu \in \mathbb{R}^3$, covariance matrix $\Sigma$, opacity $o$, and color $c$, with its spatial distribution given by:

$$G(x) = \exp\Big(-\tfrac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\Big). \quad (10)$$

In LangSplat, we augment each Gaussian with three language features $\boldsymbol{f}^s, \boldsymbol{f}^p, \boldsymbol{f}^w \in \mathbb{R}^d$, corresponding to the subpart, part, and whole semantic levels, learned in the compressed latent space. These augmented Gaussians, termed 3D language Gaussians, enable language-based querying.

To train the language features, we render them onto 2D training images using a tile-based splatting technique, which projects 3D Gaussians onto the 2D plane efficiently. The rendered language feature for pixel $v$ at semantic level $l$ is:

$$\boldsymbol{F}_t^l(v) = \sum_{i\in\mathcal{N}} \boldsymbol{f}_i^l \, \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j), \quad (11)$$

where $\mathcal{N}$ is the set of Gaussians affecting pixel $v$, $\boldsymbol{f}_i^l$ is the language feature of the $i$-th Gaussian at level $l$, and $\alpha_i = o_i G_i^{2D}(v)$ is the opacity-weighted contribution, with $G_i^{2D}(v)$ being the 2D projection of the $i$-th Gaussian.

The language features $\boldsymbol{f}_i^l$ are optimized by minimizing the loss:

$$\mathcal{L}_{\text{lang}} = \sum_{l\in\{s,p,w\}} \sum_{t=1}^{T} \sum_{v} d_{\text{lang}}\big(\boldsymbol{F}_t^l(v), \boldsymbol{H}_t^l(v)\big), \quad (12)$$

where $d_{\text{lang}}$ is a distance function (e.g., L1 or cosine distance) measuring the difference between rendered and ground truth latent features. During training, we fix the Gaussian parameters $(\mu, \Sigma, o, c)$ pre-trained for RGB rendering and only optimize the language features, ensuring consistency with the scene's visual representation.

#### 4.5.4 Open-vocabulary Querying

For inference, LangSplat supports open-vocabulary queries by rendering language features and mapping them back to the CLIP space. Given a text query with embedding $\phi_{\text{qry}} \in \mathbb{R}^D$ from the CLIP text encoder, we render the latent features $\boldsymbol{F}_t^l(v)$ for a desired view using Equation (5). These are decoded to CLIP space as:

$$\phi_{\text{img}}^l(v) = \Psi\big(\boldsymbol{F}_t^l(v)\big) \in \mathbb{R}^D.$$

The relevance score for each pixel $v$ and level $l$ is computed following LERF's strategy:

$$\text{Relevance}(v,l) = \min_i \frac{\exp\big(\phi_{\text{img}}^l(v)\cdot\phi_{\text{qry}}\big)}{\exp\big(\phi_{\text{img}}^l(v)\cdot\phi_{\text{canon}}^i\big)},$$

where $\phi_{\text{canon}}^i$ are embeddings of canonical phrases (e.g., "object," "things") to normalize scores. For each pixel,

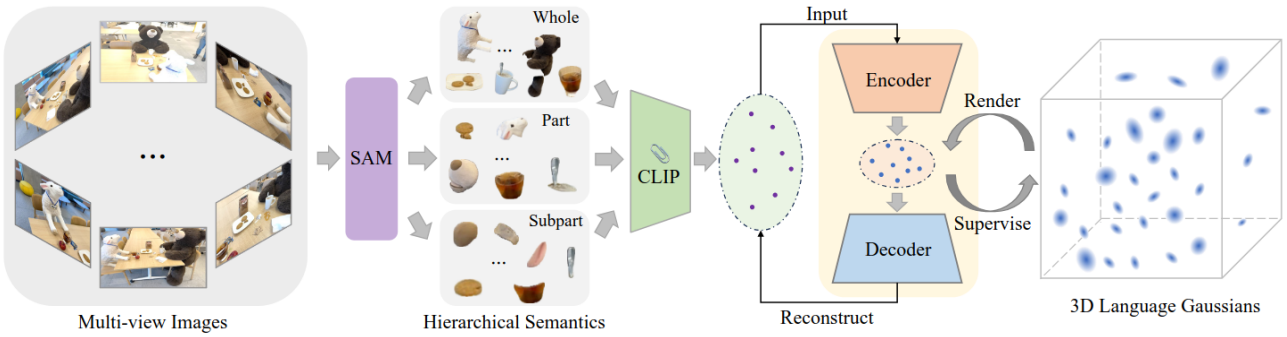



Figure 4. The framework of LangSplat. LangSplat leverages SAM to learn hierarchical semantics to address the point ambiguity issue. Then segment masks are sent to the CLIP image encoder to extract the corresponding CLIP embeddings. We learn an autoencoder with these obtained CLIP embeddings. 3D language Gaussian learn language features on the scene-specific latent space to reduce the memory cost. During querying, the rendered language embeddings are sent to the decoder to recover the features on the CLIP space.

we obtain three relevance scores and select the maximum across levels. For 3D object localization, we identify the 3D point with the highest score; for semantic segmentation, we threshold the scores to generate object masks.

## 5. Results

The results present in this report are exactly the same as those on the demo website. We present the method-wise individual results here.

### 5.1. Structure from Motion techniques

#### Incremental SfM Results

The reconstructed point cloud is visualized at various stages, allowing us to verify the incremental growth and refinement of the model. Below are sample outputs at different stages:

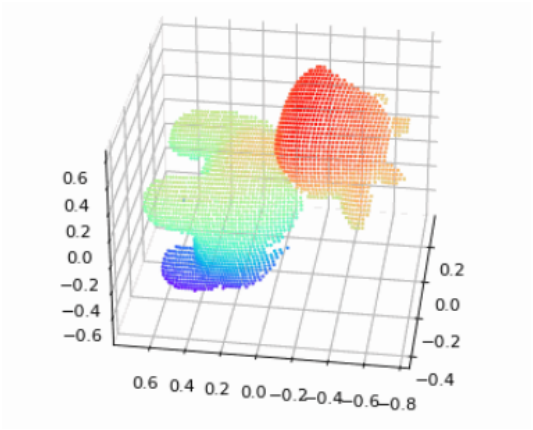


Figure 5. Reconstruction after views 1 and 2.

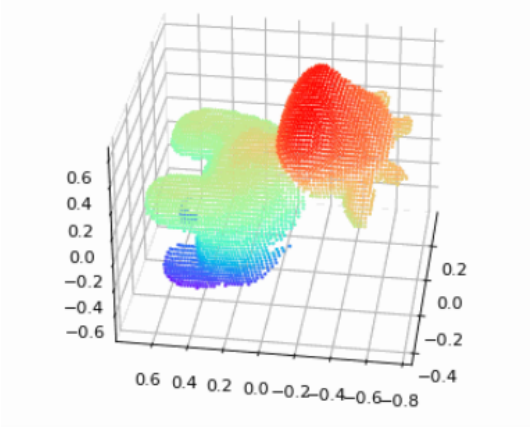


Figure 6. Expanded point cloud with view 3 integrated.

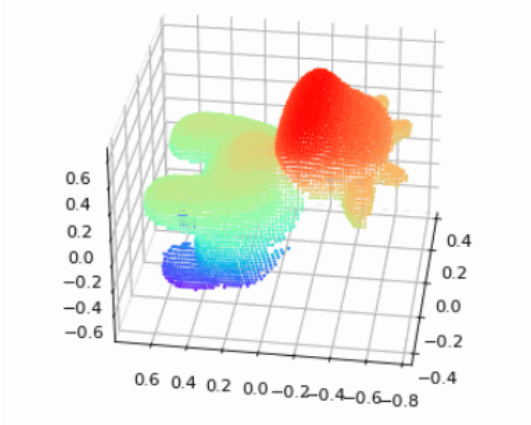


Figure 7. Final reconstruction after integrating all four views.

This step-by-step reconstruction demonstrates how new geometry is revealed progressively, and camera poses are estimated accurately using existing 3D-2D correspon-

dences.

Overall, incremental SfM is a practical and reliable method for large-scale scene reconstruction. While it may be slower than global methods, it offers flexibility and resilience in real-world conditions, making it suitable for unordered or partially corrupted image sequences.

## 5.2. Space Carving

The 3D scene reconstructions established through Space Carving methods on the Dino dataset are summarized in Figure 8. Only 2 views after reconstruction are shown for simplicity.

## 5.3. pix2Vox

Qualitative results can be found in Figure 9. For prediction using voxel-based single-view input, it takes about 2 seconds, whereas using multi-view images takes approximately 7 seconds. For the quantitative results shown in Table 3, I used the ShapeNet test dataset for both single-view and multi-view (3-view) reconstruction. I also included cases for evaluation without the context fusion network (using average fusion) and training without the discriminator (baseline).

## 5.4. Neural Radiance Field

Qualitative Results for the lego crane can be seen in Figure 10. Inference took about 5 seconds for producing 200 novel views using the Cache-based implementation.

## 5.5. Semantic Gaussian Splatting

**Results Overview.** Below, we present the results of our LangSplat reimplementation on two datasets: *LERF* and *3D-OVS*. All results are derived exclusively from the LangSplat research paper, focusing on open-vocabulary 3D semantic segmentation and 3D object localization. We also provide hardware details and report notable efficiency gains, aligning with the original paper's evaluation framework.

### 5.5.1 Dataset Overview

**LERF Dataset.** This dataset contains complex in-the-wild scenes captured via the Polycam iPhone application. It is designed for 3D object localization tasks and includes ground truth masks for open-vocabulary textual queries. Additional annotated samples, covering more challenging localization scenarios, are provided for thorough performance assessment.

**3D-OVS Dataset.** This dataset features long-tail objects in diverse poses and backgrounds, focusing on open-vocabulary 3D semantic segmentation. Its extensive set of category labels captures a wide range of real-world objects.

### 5.5.2 Hardware Details

All experiments were conducted using 2 NVIDIA A5000 GPUs.

### 5.5.3 Results on the LERF Dataset

We evaluated LangSplat for both 3D object localization and 3D semantic segmentation. Localization performance is measured by accuracy, while segmentation performance is reported using the average Intersection over Union (IoU).

See Table 4 for the 3D object localization results, and Table 5 for the 3D semantic segmentation results.
The reimplementation closely reproduces the original paper's outcomes, indicating that the hierarchical semantics extracted via SAM and the scene-specific autoencoder effectively improve open-vocabulary 3D segmentation and localization.

### 5.5.4 Results on the 3D-OVS Dataset

LangSplat's open-vocabulary 3D semantic segmentation performance on 3D-OVS is presented in Table 6. We report mean Intersection over Union (mIoU), reflecting the method's ability to accurately segment diverse object categories based solely on textual prompts.
These results underline LangSplat's robust generalization across varied object types and scene layouts, demonstrating high segmentation accuracy for long-tail categories.

# 6. Conclusion

This technical report has examined diverse 3D reconstruction methodologies, from traditional geometry-based approaches to emerging neural representations. Each paradigm offers distinct advantages with specific limitations. Traditional SfM and MVS methods provide mathematical rigor yet may falter under challenging illumination, textureless surfaces, or significant viewpoint variations, while volumetric approaches offer intuitive spatial modeling but often face resolution constraints.

Learning-based techniques, including Graph Attention Networks and GANs, harness powerful data-driven priors to achieve robust reconstructions even with incomplete or ambiguous inputs, albeit at the expense of potential overfitting and difficulty in generalizing to unseen domains. Neural implicit representations such as NeRF and Gaussian Splatting have revolutionized reconstruction fidelity, particularly in handling view synthesis and fine-detail recovery, though they demand considerable computational and memory resources.

Beyond purely geometric considerations, the integration of language understanding with 3D scene representation is an emerging frontier that facilitates semantic-level manipulation and interpretation of reconstructed scenes via

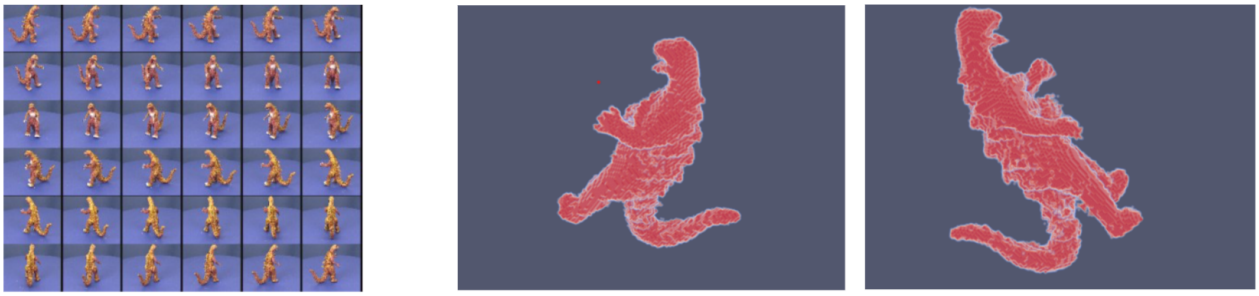



Figure 8. Input Images Grid (left) based on which the 3D Scene is reconstructed. Two sample views form the 3D scene are shown (right).

Table 3. Quantitative performance comparison across categories under different model settings. The best results are boldfaced.

| **Category** | **# Samples** | **Single-view** | **w/o Context (S)** | **w/o Adv. (S)** | **3-view** | **w/o Context (3)** | **w/o Adv. (3)** |
|---|---|---|---|---|---|---|---|
| Aeroplane | 810 | 0.690 | 0.666 | 0.684 | **0.732** | 0.724 | 0.700 |
| Bench | 364 | 0.620 | 0.602 | 0.616 | **0.673** | 0.658 | 0.647 |
| Cabinet | 315 | 0.792 | 0.789 | 0.792 | **0.822** | 0.812 | 0.812 |
| Car | 1501 | 0.857 | 0.848 | 0.854 | **0.880** | 0.875 | 0.865 |
| Chair | 1357 | 0.563 | 0.564 | 0.567 | **0.610** | 0.594 | 0.609 |
| Display | 220 | 0.534 | 0.535 | 0.537 | 0.586 | 0.576 | **0.588** |
| Lamp | 465 | 0.434 | 0.448 | 0.443 | 0.461 | 0.451 | **0.478** |
| Speaker | 325 | 0.709 | 0.716 | 0.714 | **0.746** | 0.736 | 0.745 |
| Rifle | 475 | 0.614 | 0.604 | 0.615 | **0.672** | 0.654 | 0.639 |
| Sofa | 635 | 0.708 | 0.706 | 0.709 | **0.757** | 0.742 | 0.746 |
| Table | 1703 | 0.598 | 0.598 | 0.601 | **0.635** | 0.611 | 0.625 |
| Telephone | 211 | 0.779 | 0.769 | 0.776 | **0.835** | 0.806 | 0.787 |
| Watercraft | 389 | 0.590 | 0.589 | 0.594 | **0.644** | 0.627 | 0.632 |

Table 4. Localization Accuracy (%) on the LERF Dataset.

| **Test Scene** | **LangSplat** |
|---|---|
| ramen | 73.2 |
| figurines | 80.4 |
| teatime | 88.1 |
| waldo_kitchen | 95.5 |
| **overall** | 84.3 |

Table 5. Average IoU (%) for 3D Semantic Segmentation on the LERF Dataset.

| **Test Scene** | **LangSplat** |
|---|---|
| ramen | 51.2 |
| figurines | 44.7 |
| teatime | 65.1 |
| waldo_kitchen | 44.5 |
| **overall** | 51.4 |

natural language interfaces. By marrying textual semantics with spatial reasoning, these multimodal approaches hold promise for addressing limitations of purely geometric pipelines, especially when disambiguating visually similar but semantically distinct objects.

Future developments are expected to arise from hybrid strategies that unify the precision of classical geometry with the flexibility of neural representations, bolstered by linguistic and multimodal tools. Directions for progress include:

- **Efficient Hybrid Representations:** Developing methods that combine voxel- or mesh-based geometry with learned priors, aiming for both scalable performance and high-resolution reconstructions.

- **Cross-Scene Generalization:** Refining training protocols and architectures that can handle diverse envi-

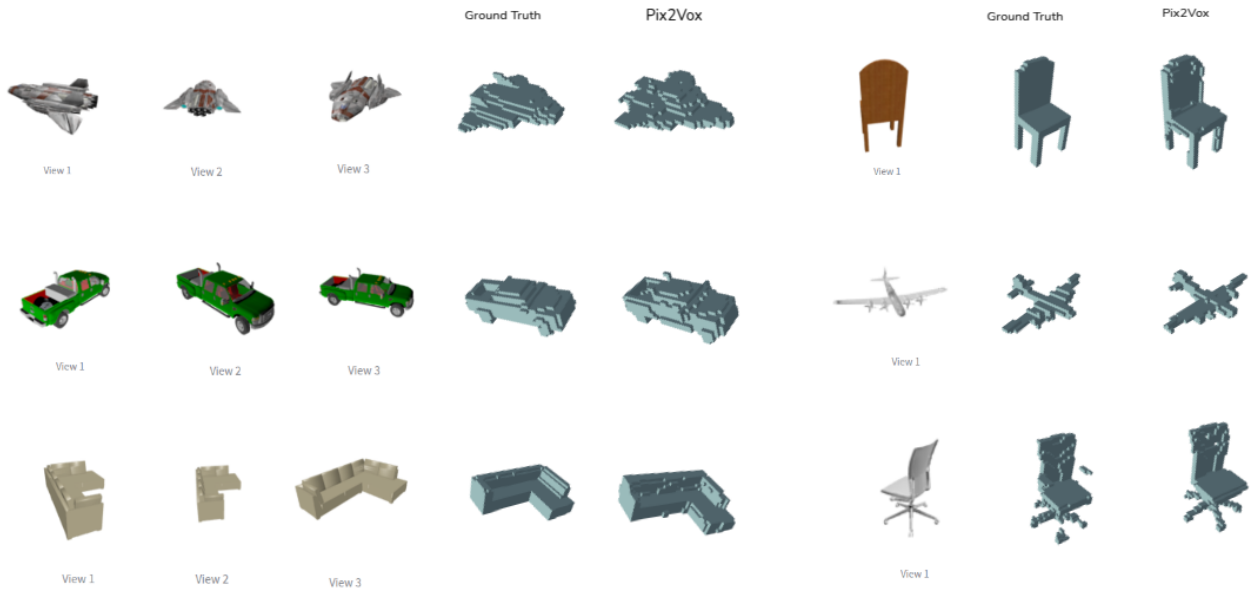



Figure 9. Multi view (left) and single view (right) reconstructions on the ShapeNet testing set.


Figure 10. Qualitative results of our implementation of FastNeRF [19], at $400^2$ pixels resolution. This also uses a medium cache whose implementation is borrowed directly from [19]. There a few sample input views provided and the goal of NeRF is to be able to synthesize a large number of novel views of the scene

Table 6. mIoU (%) for 3D Semantic Segmentation on the 3D-OVS Dataset.

| Test Scene | LangSplat |
|---|---|
| bed | 92.5 |
| bench | 94.2 |
| room | 94.1 |
| sofa | 90.0 |
| lawn | 96.1 |
| **overall** | 93.4 |

ronments without retraining or significant parameter tuning.

- **Semantic-Aware Reconstruction:** Incorporating natural language processing for scene labeling, object-level editing, and interactive scene design.
- **Uncertainty Modeling:** Quantifying ambiguity in reconstructions to highlight regions of sparse or conflicting observations.
- **Dynamic Scenes:** Extending beyond static environments to capture and represent object motion, fluid dynamics, and time-varying phenomena.

The convergence of classical geometry, deep learning, and multimodal interaction paves the way toward next-generation reconstruction systems that offer both high-fidelity modeling and semantic-level comprehension, ulti-

mately bringing us closer to an ideal pipeline with accuracy, completeness, efficiency, and remarkable adaptability.

## 7. Contributions

- Aditya Sahani - Implemented the traditioan SfM method and the Space Carving approach. Also worked on the deployment.
- Aditya Rathor - Developed the end to end pipeline for Pix2Vox along with the discriminator-based additions. Also worked on the overall deployment.
- Dishit Sharma - Worked on the implementation of the Incremental Structure from Motion methodology and deployment of this module.
- Souvik Maji - Built the Semantic Gaussian Splatting approach and the 3D Language Gaussian Splattings. Also worked on the report compilation.
- Veeraraju Elluru - Developed the Neural Radiance Fields from scratch and worked on the report compilation.

## References


[1] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3D. *ACM Transactions on Graphics*, 25(3):835–846, 2006. 1

[2] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 1

[3] C. Wu. Towards linear-time incremental structure from motion. In *3DV*, 2013. 2

[4] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010. 2

[5] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, 2016. 2

[6] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *ECCV*, 2018. 2

[7] K. Wang and S. Shen. MVDepthNet: Real-time multiview depth estimation neural network. In *3DV*, 2018. 2

[8] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, 2000. 2

[9] B. Tordoff. Carving a Dinosaur. MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/26160-carving-a-dinosaur, 2025. Retrieved April 13, 2025. 5

[10] A. Broadhurst, T. W. Drummond, and R. Cipolla. A probabilistic framework for space carving. In *ICCV*, 2001. 2

[11] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *CVPR*, 2007. 2

[12] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011. 2

[13] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *ECCV*, 2018. 2

[14] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. MeshCNN: A network with an edge. *ACM Transactions on Graphics*, 38(4):90, 2019. 2

[15] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NeurIPS*, 2016. 2

[16] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, 2016. 2

[17] J. T. Kajiya and B. P. Van Herzen.
Ray tracing volume densities.
In *Computer Graphics (SIGGRAPH)*, 1984. 8

[18] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 2, 8

[19] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. P. C. Valentin.
FastNeRF: High-fidelity neural rendering at 200FPS.
In *ICCV*, 2021. 8, 9, 14

[20] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):102, 2022. 2

[21] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *CVPR*, 2021. 2

[22] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–19, 2023. 2

[23] Y. Qin, Z. Wang, A. Zeng, Y. Du, and B. Wang. LangSplat: 3D Language Gaussian Splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2

[24] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318, 2013

[25] O. Özyeıl, V. Voroninski, R. Basri, and A. Singer. A survey of structure from motion. *Acta Numerica*, 26:305–364, 2017.

[26] J. Fuentes-Pacheco, J. R. Ascencio, and J. M. Rendón-Mancha. Visual simultaneous localization and mapping: A survey. *Artificial Intelligence Review*, 43(1):55–81, 2015.

[27] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint arXiv:1512.03012*, 2015. 6

[28] A. Kar, C. Häne, and J. Malik. Learning a multi-view stereo machine. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 365–376, 2017.

[29] M. Wang, L. Wang, and Y. Fang. 3DensiNet: A robust neural network architecture towards 3D volumetric object prediction from 2D image. *Proceedings of the ACM International Conference on Multimedia (ACM MM)*, pages 790–798, 2017.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 6

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015. 6

[32] H. Xie, H. Yao, S. Sun, X. Zhang, and S. Tian. Pix2Vox: Context-aware 3D reconstruction from single and multi-view images. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2690–2698, 2019.

[33] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015. 6

[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2672–2680, 2014. 6

[35] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

## A. WebGL Viewer for Gaussian Splatting

This section documents the custom WebGL-based viewer developed for real-time visualization of Gaussian Splatting. The viewer offers interactive navigation and supports a wide range of input modes for both desktop and mobile platforms.

### Controls Overview

**Movement (Arrow Keys)**

- **Left / Right**: Strafe side to side
- **Up / Down**: Move forward/back
- **Space**: Jump

**Camera Angle (WASD + Others)**

- **A / D**: Turn camera left/right
- **W / S**: Tilt camera up/down
- **Q / E**: Roll camera counterclockwise/clockwise
- **I / K and J / L**: Orbit vertically and horizontally

**Trackpad**

- **Scroll**: Orbit
- **Pinch**: Move forward/back
- **Ctrl + Scroll**: Move forward/back
- **Shift + Scroll**: Move up/down or strafe

**Mouse**

- **Click and drag**: Orbit
- **Right click (or Ctrl/Cmd + drag)**: Move

**Touch (Mobile)**

- **One finger**: Orbit
- **Two finger pinch**: Move forward/back
- **Two finger rotate**: Rotate camera
- **Two finger pan**: Move side-to-side and up/down

**Additional Controls**

- **0–9**: Switch to pre-loaded camera views
- **- / +**: Cycle loaded cameras
- **P**: Resume default animation
- **Drag & drop .ply file**: Convert to .splat
- **Drag & drop cameras.json**: Load cameras


Figure 11. WebGL Gaussian Splatting Viewer