Exo-Centric to Ego-Centric View Translation using Gaussian Splatting

Yogeshwar Boopathy

Student ID: 2723440

A dissertation submitted for the degree of

Master of Science in Data Science and Artificial Intelligence

Supervised by *Dr. Hyung Jin Chang*

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

2023-24

Abstract

This project explores the application of Gaussian Splatting techniques to facilitate the translation of exocentric (third-person) views into egocentric (first-person) perspectives, addressing the significant challenges posed by transforming a distant observer's viewpoint into an immersive first-person perspective. This translation has important implications in fields such as robotics, virtual reality (VR), augmented reality (AR), and human-computer interaction, where understanding how actions appear from the actor's viewpoint is essential.The primary aim of this project is to enhance the accuracy, quality, and realism of these view translations. By using Gaussian Splatting, a cutting-edge rasterization technique, the project achieves the rendering of photorealistic scenes using sparse image datasets. This method allows for the efficient representation of scenes through Gaussian points in 3D space, enabling detailed and realistic outputs, particularly in scenarios involving complex hand-object interactions.Integrating Gaussian Splatting into the exo-to-ego view translation pipeline significantly improves the process, reducing distortions, minimizing occlusion issues, and preserving finer details that are often lost in traditional methods. The performance of the proposed approach is validated using quantitative metrics such as Structural Similarity Index (SSIM) and Peak Signal-to-Noise Ratio (PSNR) on real-world datasets, demonstrating notable improvements over conventional techniques.In conclusion, this project advances the technical capabilities of exocentric to egocentric view translation through the use of Gaussian Splatting. The results highlight its potential to deliver more accurate, high-quality, and real-time ego-centric view generation, opening new possibilities for immersive experiences across various interactive domains.

Acknowledgements

I would first like to thank my project supervisor, Dr. Hyung Jin Chang, for his guidance and support throughout this project.

I would like to thank my friend and Doctoral Researcher, Yuqi Hou, for providing me with the necessary computing resources, ideas and encouragement throughout this project.

I would like to thank my project inspector, Dr. Venelin Kovatchev for his valuable feedback and suggestions.

I would like to thank my friends Shashwat, Aasiya, and Pradyum for volunteering to be a part of the dataset and my friend Sharief for filming a scene in the dataset.

And finally, I would like to thank my family and friends for their support and encouragement throughout this year and the writing of this dissertation.

Abbreviations

GAN	Generative Adversarial Network
P-GAN	Parallel Generative Adversarial Network
NeRF	Neural Radiance Fields
MSE	Mean Squared Error
SRN	Scene Representation Networks
VAE	Variational Auto Encoder
SfM	Structure from Motion
SIFT	Scale Invariant Feature Transform
ORB	Oriented FAST and Rotated BRIEF
MVS	Multi-View Stereo
RANSAC	Random Sample Consensus
3dGS	3D Gaussian Splatting
FOV	Field of View
MLP	Multi-Layer Perceptron
DyNerf	Dynamic Neural Radiance Fields
SSIM	Structural Similarity Index
PSNR	Peak Signal-to-Noise Ratio
LPIPS	Learned Perceptual Image Patch Similarity
PDF	Probability Density Function
VR	Virtual Reality
AR	Augmented Reality

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and Motivation

In computer vision and artificial intelligence research, ego-centric (first-person) vision has gained significant attention due to the growing popularity of wearable cameras like GoPros and other first-person devices. These cameras provide a distinct viewpoint that shows the world from an individual's perspective, which is particularly helpful for comprehending a thorough first-person view. In various fields, such as robotics, immersive media, augmented reality, and virtual reality, ego-centric vision has become increasingly important. Ego-centric cameras record interactions with far more detail than exo-centric (third-person) cameras, giving viewers a closer look at the subtleties of human behaviour and a representation of the activities from the subject's perspective.

Converting exocentric viewpoints into egocentric ones is one of the trickiest problems in ego-centric vision research. Though much has been accomplished in computer vision, little has been done to connect these two viewpoints. Mirror neurones enable humans to effortlessly imagine the visual world from another person's perspective, mainly while manipulating objects. However, because of the sharp differences in spatial relations and occlusion between the two viewpoints, existing computer vision algorithms struggle with this translation. First-person (ego-centric) views offer a more in-depth, smaller perspective that concentrates on hand movements and items that are directly within the user's field of vision, whereas third-person (exocentric) views offer a wider angle of observation over scenes. This difference makes creating an accurate and realistic egocentric perspective more difficult.

While existing cross-view image-generating algorithms have achieved good results in analogous tasks, they are ineffective in solving the exo-to-ego translation challenge. For instance, generative models, such as Generative Adversarial Networks (GANs), have been applied to various image synthesis tasks. Still, they are not well suited to handle the notable distinctions between ego-centric and exocentric viewpoints. Conventional methods for achieving vision synthesis depend on geometric models or camera characteristics, which are frequently computationally expensive and don't translate well into real-world situations. Furthermore, cross-view translation techniques like CycleGAN [22] and Pix2Pix [4] have demonstrated potential in creating paired and unpaired images. However, exo-to-ego translation shows that they are constrained when views have minimal overlapping information.

This work is motivated by the urge to develop a more robust solution to this under-explored problem of exo-to-ego translation. Whereas there have been tremendous works in 3D scene reconstruction and neural rendering, few works have gone into actual exo-to-ego translation. This project overcomes limitations presented by traditional methods through Gaussian Splatting [5], a potent technique in novel view synthesis. Gaussian Splatting allows the rendering of scenes as a set of Gaussian distributions, thereby allowing for elastic handling of complex dynamic environments. Besides that, it has excellent potential to enhance translation accuracy and quality of exocentric-to-egocentric views. Thus, it is very promising for improving immersive experiences within both VR/AR and other interaction applications.

1.2 Aims and Objectives

The main aim behind this project is to devise a method that will convert exocentric, third-person views into egocentric (first-person perspective) using Gaussian Splatting, one of the state-of-the-art real-time techniques for scene rendering. Translation of such two different perspectives is indispensable in many applications related to computer vision, robotics, virtual and augmented realities, and human-computer interaction. In human perception, these two perspectives will quickly melt into each other due to the innate neural mechanisms of mirror neurons. On the other hand, computer vision systems face tremendous challenges posed by significant differences in viewpoint, spatial relationships, and occlusions. The paper discusses this challenge and proposes one computational framework that uses Gaussian Splatting to realize an accurate view translation between these two perspectives with high quality. The main objectives of the project are highlighted below:

- **Investigate Current Methods for Cross-View Translation:** This will be a careful review of current techniques that exist for exocentric-to-egocentric view translation, considering classical computer vision approaches, such as geometry-based models and neural rendering methods, for example, NeRF, and generative models using GAN. The emphasis will be on reviewing strengths and limitations, considering challenging scenarios involving hand-object interactions.
- **Develop a Gaussian Splatting-Based Framework for View Translation:** The main objective of this project is to contribute to and implement a novel framework based on Gaussian Splatting for translating exocentric views into ego-centric perspectives. Gaussian Splatting can be defined as the representation of a scene as Gaussian distributions. It allows for high efficiency and flexibility in rendering dynamic scenes. This contribution will optimize the representation of 3D spaces and enable more accurate and photo-realistic ego-centric view synthesis.
- **Create a Dataset of Real-World Scenes for View Translation:** One of the prime objectives of this thesis is to collect and develop a dataset composed of real-world scenes, particularly for exocentric-to-egocentric view translation. This should include a wide range of activities and interactions relevant to ego-centric views that involve hand-object manipulations and complex real-world movements to test the model in realistic conditions.
- **Optimize the Algorithm for Ego-centric Views:** Refine the Gaussian Splatting-based algorithm to be optimized concerning the peculiar demands imposed by ego-centric displays. This will involve model parameter tuning and rendering strategy adaptation to focus on fine-grained, indispensable details given first-person perspectives.
- **Evaluate Performance Using Quantitative Metrics:** After developing the framework, this would be evaluated by quantitative metrics such as SSIM, PSNR, etc. A comparison with the state-of-the-art methods will be performed to show the improvements and emphasize the advantages of using Gaussian Splatting for the task above.

1.3 Challenges

- **Significant Viewpoint Discrepancy:** The significant changes in exocentric and ego-centric perspectives make it hard to accurately model view relationships and appearances, which challenges the existing approaches of synthesizing views.
- **Handling Complex Motions:** Dynamic scene capture and reconstruction, particularly with delicate hand-object interactions, require advanced models to efficiently cope with spatial and temporal complications.
- **Occlusion and Ambiguity:** This would involve extrapolating occluded objects or unseen parts of the scene, which requires a model to infer and generate realistic views based on incomplete data—a very challenging task for the current crop of methods.
- **Generalization to Real-World Scenarios:** Most existing models can not generalize to real-world scenarios of diverse objects, subjects, and backgrounds with which they are confronted due to their limited training. Therefore, more robust and adaptive methods will be required.

1.4 Thesis Structure

Chapter 1 motivates the project and sets the background for view translation in the context of computer vision. The chapter also covers the aim and objectives of the research. Finally, it gives an overview of the thesis structure to guide the reader through the research process.

Chapter 2 provides a critical review of the related literature, covering essential techniques and methodologies related to cross-view translation. Specific topics include traditional geometry-based models, neural rendering methods like NeRF, and state-of-the-art P-GAN and generative-based approaches. This chapter identifies gaps in existing research and explains how this project contributes to the knowledge base by using Gaussian Splatting to solve the view translation problem.

Chapter 3 describes the methodology and framework adopted in this project. The chapter discusses about the design and implementation of Gaussian Splatting for view translation, thereby highlighting its advantages in real-time scene rendering and its applicability to cross-view synthesis. It also details the steps taken in dataset curation, algorithm optimization for ego-centric views, and the definition of evaluation metrics. The chapter provides a step-by-step explanation of the development process.

Chapter 4 presents the experiments and results. It offers insights into the proposed system's quantitative and qualitative performance. This chapter also compares the method's performance against traditional techniques and highlights key findings that demonstrate the efficacy of Gaussian Splatting for exocentric-to-egocentric view translation.

Chapter 5 summarizes the key contributions and findings of the research. It reflects on the overall impact of applying Gaussian Splatting to the view translation problem and outlines the limitations encountered during the project. This chapter also provides recommendations for future research, including algorithm improvements, expanding the dataset, and exploring other real-world applications for ego-centric view translation where broader benefits may be realized.

CHAPTER 2

Related Work

The field of view translation, specifically the transition from exocentric (third-person) to egocentric (first-person) perspectives, is a relatively new but rapidly evolving area within computer vision. Prior work has explored the complexities of translating views with significantly different viewpoints, particularly the challenges associated with significant viewpoint shifts, occlusions, and maintaining high-fidelity details in generated images. Various techniques, ranging from traditional geometric models to advanced neural networks like Parallel Generative Adversarial Networks (GANs) and others, have been applied to tackle these issues. However, many existing approaches struggle with the drastic difference between exocentric and egocentric perspectives, which present unique visual and spatial consistency. challenges This chapter will highlight studies that have addressed similar challenges, including the use of generative models, neural rendering, and geometric approaches for cross-view image synthesis.

2.1 Parallel Generative Adversarial Network

The **Parallel Generative Adversarial Network** (P-GAN) architecture as shown in Fig. 2.1 from the paper 'Parallel Generative Adversarial Network for Third Person to First Person Image Generation' (Liu et al., 2020) [6] handles the task of exocentric-to-egocentric view translation by employing two separate GANs in parallel—one to each view. These GANs share a common set of layers in the early stages of the network. Specifically, the first three layers of the two generators are hard-shared, meaning they have identical structures and weights. This shared portion of the network is responsible for learning high-level semantic information common between exo-centric and egocentric views, such as object shapes, spatial relationships, and general scene structure. After these shared layers, the network splits into view-specific pathways to capture the unique aspects of each perspective.

A **U-Net** architecture is utilized for the generators. U-Net is known for handling image translation tasks by preserving high-resolution information through skip connections between the down-sampling and up-sampling paths. This helps maintain fine details in the generated images. The discriminators used are PatchGANs, which evaluated the local patches of the image rather than the entire image. This approach allows the network to focus on local texture realism, which is particularly useful for tasks like view translation, where fine details like hand gestures and object texture must be accurate.

2.1.1 Architecture and Working of P-GAN

P-GAN utilizes two parallel GANs, one for each view—one GAN is responsible for synthesizing the egocentric view from an exocentric input, and the other GAN synthesizes the exocentric view from an egocentric input. The architecture of these two GANs shares a portion of layers (referred to as hard-sharing of layers) to learn common high-level features between the views. This shared structure is crucial because the egocentric and exocentric views contain shared semantic information, such as object shapes, spatial configurations, and relationships within the scene.

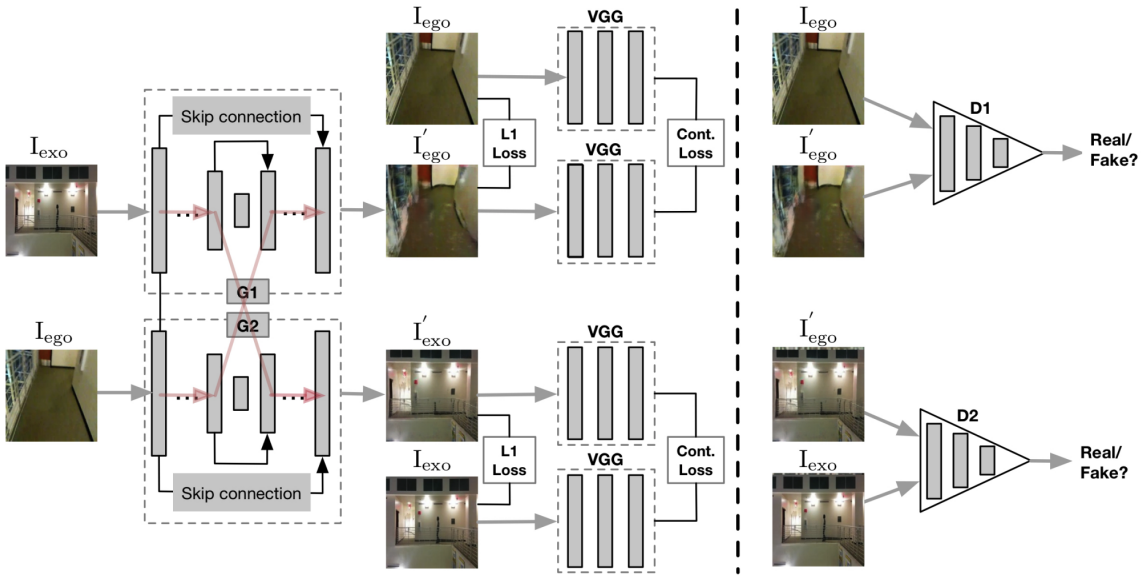

Figure 2.1: Pipeline of Parallel-GAN

Each GAN has a **generator** and **discriminator**. The role of the generator is to synthesize images in the target view (either egocentric or exocentric) from the source view. In P-GAN, the generator architecture is based on the U-Net model. U-Net has an encoder-decoder structure where the encoder captures high-level features, and the decoder reconstructs the image in the target domain. U-Net also uses skip connections, which transfer fine-grained details from the encoder to the decoder, helping to preserve critical image information. The discriminator evaluates the authenticity of the generated image, determining whether it is real or synthesized. P-GAN uses a PatchGAN discriminator, which divides the image into smaller patches and evaluates the realism of each patch individually. This helps the model focus on fine-grained textures, and local details are important for tasks like view translation.

A unique feature of P-GAN is the **hard-sharing of the initial layers** in the two generators. The first three layers of both generators are shared across both tasks—egocentric-to-exocentric and exocentric-to-egocentric translations. This means that the model learns to extract common features across both perspectives. These shared layers are responsible for encoding high-level semantics, such as object shapes, sizes, and spatial layouts consistent across both views. Once the shared layers have processed the input, the model then branches into separate pathways that handle view-specific transformations. This is essential because both views share some high-level information, but they also contain unique details that the view-specific branches must capture.

To ensure that the translations between views are consistent and reversible, P-GAN incorporates a **cross-cycle loss**. This loss function enforces the following procedure:

- Given an exocentric image (third-person view), the generator translates it into an egocentric image (first-person view).
- The generated egocentric image is then fed back into the second generator to produce an exocentric image.
- The resulting exocentric image should match the original exocentric image. This process is also performed in the reverse direction.

The cross-cycle loss ensures that the translations between the two views are consistent, i.e., if one translates from exocentric to egocentric and then back to exocentric, the result should closely resemble the original input. This consistency constraint helps the model generate more accurate and realistic translations between the two perspectives.

One of the most important components of P-GAN is the **contextual feature loss**. This loss is based on the feature maps extracted from a pre-trained VGG-19 network. VGG-19 is a deep convolutional

neural network trained on a large image dataset (ImageNet), and it is known for its ability to capture rich, hierarchical features from images. In P-GAN, the contextual feature loss ensures that the textures, spatial relationships, and object details are preserved in the generated images. For example, suppose the model is translating an exocentric view where a person is holding an object. In that case, the contextual feature loss ensures that the textures and fine-grained details of both the hand and the object are preserved when translated into the egocentric view.

As with any GAN, P-GAN utilizes an **adversarial loss**. This loss comes from the competition between the generator and the discriminator:

- The generator tries to synthesize images that are indistinguishable from real images in the target view.
- The discriminator tries to distinguish between real images and the images generated by the model.

The adversarial loss helps the generator create high-quality, realistic images. In the context of P-GAN, adversarial loss ensures that the generated egocentric or exocentric images are not only semantically correct but also realistic to a human observer. The PatchGAN discriminator plays a key role here by focusing on the local textures and details, helping to prevent the generation of blurry or unrealistic images.

In addition to the adversarial loss and cross-cycle loss, P-GAN also incorporates a **reconstruction loss**. This loss ensures that the generated image, when translated back to the original view, closely resembles the original image. This further enforces the consistency of the generated images and prevents the model from generating images that deviate too much from the source view.

2.1.2 Key Highlights of P-GAN

- **Parallel GAN Structure:** The introduction of two parallel GANs allows P-GAN to learn from both views simultaneously. This design helps in capturing the semantic relationship between exocentric and egocentric views more effectively, making the translation between these perspectives more seamless.
- **Hard-Sharing of Layers:** Sharing the first three layers of the two generators reduces computational redundancy by learning common high-level features (e.g., object shapes and spatial layout) that exist in both views. This not only speeds up training but also helps the model generalize better across different scenes, reducing the risk of overfitting to one particular view.
- **Cross-Cycle Loss:** This loss ensures that the transformations between exocentric and egocentric views are consistent. It guarantees that if an image is translated from one view to another and then back, the resulting image closely matches the original. This enforces cycle consistency, which is key to generating realistic images that preserve the essential characteristics of the scene.
- **Contextual Feature Loss:** Extracted from the VGG-19 network, this loss function ensures that the generated images maintain fine details, such as textures and spatial relationships, which are essential for high-quality view translation. This is particularly important in tasks that involve close interactions between hands and objects, where fine-grained details are crucial for realism.
- **PatchGAN Discriminators:** These discriminators focus on the local texture and pattern realism by evaluating small patches of the image. This approach ensures that the fine details in each region of the generated image are realistic, improving the overall quality of the translation. The use of PatchGAN helps avoid common artifacts that can appear in GAN-generated images, such as blurry textures or unrealistic object boundaries.

2.1.3 Limitations of P-GAN

- **Challenges with Complex Interactions:** While P-GAN performs well in simpler scenarios like walking or running, it faces challenges when dealing with complex hand-object interactions. Activities such as assembling toys or pouring liquid, which involve intricate hand movements and object manipulations, are harder to translate due to the inherent complexity and frequent occlusions that occur between the hands and the objects.

- **Difficulty with Large Viewpoint Changes:** The model struggles with significant viewpoint shifts, such as translating from a wide, distant exocentric view to a close-up egocentric view. The lack of overlap in visual content between these two perspectives makes it difficult for the model to accurately synthesize the correct egocentric image, especially when critical information (like hand movements or small objects) is not visible in the exocentric view.
- **Data Dependency:** The success of P-GAN is highly dependent on the availability of paired exocentric and egocentric datasets, which are often challenging to collect in real-world settings. Without sufficient training data, the model may fail to generalize to new environments, limiting its practical applicability.
- **Computational Complexity:** P-GAN's architecture, with its dual-GAN structure and multiple loss functions (e.g., cross-cycle, contextual feature, adversarial losses), results in a high computational cost. This makes the model challenging to deploy in real-time applications or environments with limited computational resources, such as mobile devices or edge computing scenarios.

Geometry Aware Novel View Synthesis

2.2 Neural Radiance Fields

The Exo-to-Ego view also relates to Novel view synthesis. **Neural Radiance Fields (NeRF)** [9] is a groundbreaking model for 3D scene representation that can synthesize novel views of a scene given a set of 2D images. NeRF represents a continuous 3D scene as a neural network that outputs color and density values for any given point in 3D space. This allows the model to render photorealistic views from novel viewpoints through a process known as differentiable neural rendering.

2.2.1 Architecture of NeRF

NeRF represents a scene using a fully connected neural network that maps a 5D input (spatial coordinates and viewing direction) to 4D outputs (color and volumetric density) shown in Fig. 2.2 The NeRF takes as input the following :

- 3D coordinates: $\mathbf{x} = (x, y, z)$ represent the spatial location of a point in the scene.
- Viewing direction: $\mathbf{d} = (\theta, \phi)$ represents the direction from which this point is being viewed, typically using spherical coordinates.

A fully connected deep neural network processes the input. The network consists of several layers (typically 8 or more), with each layer applying a non-linear activation (usually ReLU) to produce intermediate feature vectors. The spatial coordinates $\mathbf{x}$ are processed by the network to produce an intermediate feature vector, from which the volumetric density σ (a scalar value representing how much light is absorbed at that point) is predicted. To better capture fine details and high-frequency information, NeRF applies a positional encoding to both the 3D coordinates and the viewing direction. This encoding maps the coordinates into a higher-dimensional space using a series of sinusoidal functions.

After predicting the volumetric density σ, the intermediate feature vector is concatenated with the viewing direction $\mathbf{d}$, and the combined information is passed through additional layers to predict the RGB color $c = (r, g, b)$ at that point. The final output is a radiance field: a continuous 3D representation of the scene, where each 3D point has an associated density σ and color c.

2.2.2 Working of NeRF

NeRF's core principle is to represent a scene as a **continuous function** that can be queried at any 3D point to determine the color and density of the scene at that location. It works in conjunction with **volume rendering** to produce novel views from arbitrary camera angles.

NeRF is trained on a collection of 2D images of the scene captured from multiple viewpoints, along with their associated camera poses (intrinsic and extrinsic parameters). These images are typically captured in a 360-degree manner around the object or scene.

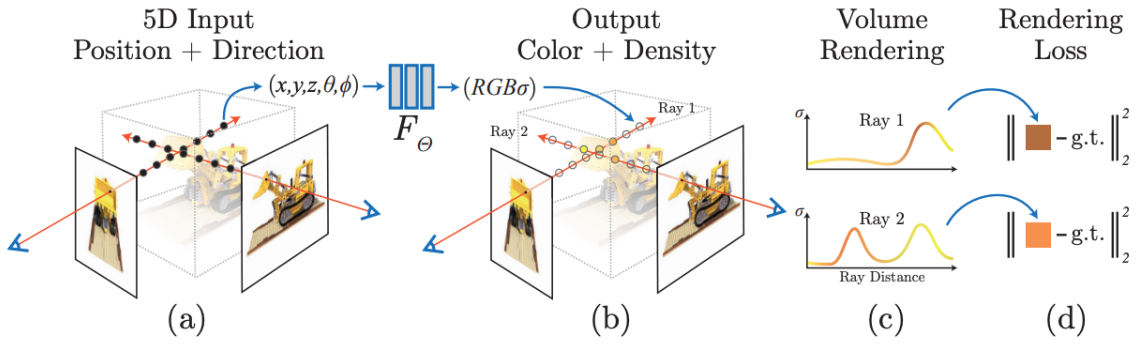

Figure 2.2: Neural Radiance Fields

For each pixel in the target image (a new view to be rendered), NeRF casts a ray from the camera center through the pixel into the 3D scene. The direction of this ray is defined by the camera parameters and the pixel's position in the image. The ray is sampled at multiple points $\mathbf{x}_i$ along its path. These sample points represent locations in 3D space where the scene could potentially contribute to the color of the corresponding pixel.

Point Sampling and Network Query: For each sample point $\mathbf{x}_i$, NeRF queries the neural network with the 3D coordinates of the point and the viewing direction. The network outputs the color $c_i = (r_i, g_i, b_i)$ and volumetric density σ_i for that point.

The colors and densities along the ray are combined using **volume rendering** to compute the final color of the pixel. This is done by accumulating the colors along the ray, weighted by their densities (to simulate the absorption and emission of light as it passes through the scene). The pixel's final color $C(\mathbf{r})$ is calculated using the following integral:

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

where $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(s)ds\right)$ is the transmittance, which represents the fraction of light that reaches the point without being absorbed.

NeRF is trained by comparing the rendered pixels to the ground-truth pixels from the input images. The loss function typically used is the **mean squared error (MSE)** between the predicted and actual pixel colors. Over the course of training, the network learns to represent the scene's geometry and appearance in a way that allows it to generalize to new viewpoints.

2.2.3 Key Highlights of NeRF

- **High-Quality Novel View Synthesis:** NeRF can generate photorealistic novel views of a scene, generally producing very high-quality results for smooth surfaces and fine details.
- **Continuous 3D Representation:** NeRF represents a scene continuously over time, while voxel-based representations discretize the space. This allows for more detailed and memory-efficient modeling of complex geometries.
- **Differentiable Rendering:** NeRF employs differentiable volume rendering that allows for end-to-end training. This helps the model optimize both the scene geometry and its appearance based on the input images.
- **Positional Encoding:** By using positional encoding, NeRF effectively models both high-frequency details, like textures, and smooth surfaces. This technique helps to overcome the intrinsic limitations of neural networks in representing high-frequency variations.

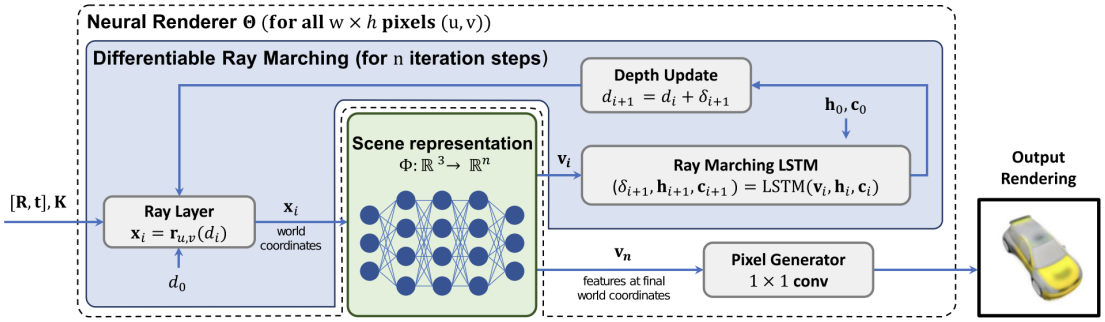

Figure 2.3: Scene Representation Network

2.2.4 Limitations of NeRF

While NeRF has demonstrated remarkable results in novel view synthesis, it faces several challenges when applied to the exo-to-ego view translation task:

- **Reliance on Dense Viewpoints:** NeRF is typically effective because it is trained with a large number of images taken from multiple viewpoints around the scene. However, in exo-to-ego view synthesis, the availability of viewpoints is sparse and not well-distributed. This reduces NeRF's capacity to handle extreme view changes, such as transitioning from an exocentric view to an egocentric one.
- **Limited Extrapolation Capability:** NeRF excels at interpolating between known views but performs poorly when it comes to multi-view extrapolation. For exo-to-ego translation, large viewpoint changes, like going from a third-person (exo) view to a first-person (ego) view, require the model to synthesize parts of the scene that were never visible in the input images.
- **Camera Pose Dependency:** NeRF requires precise camera parameters (intrinsic and extrinsic) for every input image. In real-world exo-to-ego scenarios, such camera parameters are often unavailable or difficult to infer accurately from the exocentric video frames. This makes it challenging to apply NeRF directly to exo-to-ego view synthesis, where camera poses may not be reliable or provided at all.
- **Occlusion and Ambiguity:** In exo-to-ego translation, parts of the scene, such as hands or objects in the ego view, may be occluded in the exo view. NeRF, trained with regression objectives, is not well-equipped to handle occlusion or ambiguity and often fails to generate the occluded parts of the scene. Generative models, with their stochastic nature and priors, tend to handle this challenge better.
- **Computational Cost:** NeRF incurs significant computational costs in terms of both memory and rendering time. Volume rendering requires evaluating the network at multiple points along each ray for every pixel, leading to extended training and inference times. This poses challenges for real-time applications, like exo-to-ego view synthesis in interactive environments.

2.3 Scene Representation Networks

Scene Representation Networks (SRNs) [14], are a class of models designed for 3D scene representation and novel view synthesis. Unlike traditional voxel-based 3D models or point clouds, SRNs represent scenes implicitly as continuous neural functions. These neural functions map 3D coordinates to scene properties (e.g., color and density) without explicitly storing the 3D geometry, thus enabling the reconstruction of complex 3D scenes from 2D images.

2.3.1 Architecture of SRN

Scene Representation Networks (SRNs) as shown in Fig. 2.3 use an **implicit**, **continuous neural representation** to model 3D scenes and generate novel views. The architecture of SRNs revolves around two key components: an implicit scene representation and a differentiable rendering function. Together, these components allow the model to synthesize 2D images from any given camera viewpoint without requiring explicit 3D data or discrete geometry representations such as voxels or meshes.

The implicit scene representation is realized through a fully connected neural network that takes a 3D coordinate, x = (x, y, z), as input. This coordinate represents a point in 3D space. Instead of directly mapping the coordinate to RGB values (color), the network produces a high-dimensional feature vector for that point. This feature vector encodes the local properties of the scene at that coordinate, such as appearance, geometry, and texture. This feature field serves as a continuous 3D representation of the scene, meaning the entire scene is implicitly stored in the parameters of the neural network rather than in a grid of 3D points or other explicit structures.

Next, the architecture employs a **differentiable rendering function** to transform the latent 3D feature vectors into 2D images from arbitrary camera viewpoints. This process relies on a technique called ray marching, where rays are cast from the camera into the scene. For each pixel in the 2D image, a ray is traced through the 3D scene, and the neural network evaluates the feature field at multiple points along the ray. The features sampled along the ray are integrated to compute the final pixel color and opacity. This rendering process is fully differentiable, meaning gradients can flow through the entire system, enabling end-to-end network training using image-level supervision.

The camera poses for the input images are crucial in this architecture. The camera parameters (both intrinsic and extrinsic) are needed to define the direction and origin of the rays that are cast into the scene. SRNs use these camera poses to sample the 3D feature field, which allows the model to render novel views from previously unseen perspectives.

2.3.2 Working of SRN

The working of SRNs begins with the input images and corresponding camera poses. These are used to train the neural network to map 3D points in space to a latent feature representation. During training, the network takes 3D coordinates as input and outputs a feature vector for each point, encoding information about the local scene structure and appearance at that location. The goal is to have the network learn a continuous function that represents the scene, such that it can generate realistic images when queried from arbitrary viewpoints.

Once the 3D scene is encoded in this implicit feature field, the network can render 2D images by simulating the process of taking a photograph from any virtual camera position. To render a novel view, rays are cast from the camera through each pixel in the image plane and into the 3D scene. The neural network evaluates the feature field at several points along each ray, collecting information about the scene's appearance at those points. These features are then combined using a process similar to volume rendering, where the features along the ray are integrated to produce the final pixel color.

The differentiable nature of this rendering process is key to the SRN's ability to learn from images alone. During training, the network's predictions are compared to the ground-truth images, and the error is propagated back through the network. By adjusting its parameters to minimize this error, the network gradually learns a representation of the scene that can be used to generate realistic images from new viewpoints. This learning process leverages the fact that the network can not only store complex 3D information in its weights but also render 2D images in a manner that is smooth and continuous across viewpoints.

Importantly, SRNs are not limited by fixed resolution 3D representations like voxel grids. Instead, they can represent the scene at any level of detail, as the 3D scene is implicitly defined by the neural network function rather than by a discrete set of points or voxels. This continuous representation allows the network to render detailed images and perform tasks such as view interpolation, where it generates images from viewpoints between those provided during training.

In summary, SRNs work by learning a mapping from 3D coordinates to scene properties through a neural network, which implicitly stores the 3D geometry and appearance of a scene. A differentiable rendering function then allows the model to simulate the process of capturing an image from any camera viewpoint, using the neural network to predict the color and opacity of each pixel based on the scene's latent representation. This approach allows SRNs to generate novel views and perform tasks like 3D

reconstruction without relying on explicit 3D data.

2.3.3 Key Highlights of SRN

- **Implicit 3D Representation:** SRNs represent the entire continuous 3D scene using a neural network, avoiding the need for explicit voxel grids or point clouds.
- **Differentiable Rendering:** The model renders in a differentiable way by marching rays, enabling end-to-end training using only 2D image data.
- **High-Quality Novel View Synthesis:** SRNs can synthesize realistic novel views from arbitrary camera viewpoints by reprojecting learned features of the underlying scene.
- **Resolution Independence:** Due to the implicit nature of SRNs, flexible scene representation is possible without being constrained by the fixed resolution of voxel grids.

2.3.4 Limitations of SRN

- **Camera Pose Dependency:** SRNs rely on precise camera pose information, which might not always be available in real-world scenarios.
- **Limited Scalability to Complex Scenes:** SRNs struggle to efficiently represent large, complex scenes that contain high geometric and appearance variation.
- **Handling of Occlusion:** SRNs may face challenges with occluded regions and sparse input images, leading to ambiguity when rendering occluded parts.

2.4 Generative Based Framework for View Synthesis

The paper titled 'Put Myself in Your Shoes: Lifting the Egocentric Perspective from Exocentric Videos' (Luo et al., 2024) [8] proposed a generative framework called Exo2Ego as shown in Fig. 2.4 that decouples the translation process into two stages: **high-level structure transformation**, which explicitly encourages cross-view correspondence between exocentric and egocentric views, and a **diffusion-based pixel-level hallucination**, which incorporates a hand layout prior to enhance the fidelity of the generated egocentric view.

2.4.1 Architecture and Working of Exo2Ego Framework

The **Exo2Ego** framework for exo-to-ego view translation is designed to handle the challenging task of converting an exocentric view into an egocentric one. This framework consists of two main components: **High-level Structure Transformation** and **Diffusion-based Pixel Hallucination**, each playing a specific role in the process of transforming exocentric video frames into egocentric ones. Here's a detailed explanation of each component using LaTeX notation for the key variables:

High-level Structure Transformation

This component is responsible for predicting the approximate hand-object interaction structure in the egocentric view from an exocentric frame. To achieve this, a transformer-based encoder-decoder architecture is employed.

Exo Contextual Feature Extraction (Encoder): It inputs an exocentric video frame $X_t \in R^{H\times W\times C_1}$, where H, W, and C_1 represent the height, width, and number of channels (typically 3 for RGB), along with a corresponding exocentric layout $L_t^x \in R^{H\times W\times C_2}$, which provides the 2D hand pose layout for the exo frame.

The exocentric frame X_t and layout L_t^x are divided into smaller patches, similar to how Vision Transformers (ViTs) operate. Each patch is flattened and embedded into a token, forming a sequence. These tokens are augmented with position embeddings to retain spatial information. The token sequence is passed through L layers of a **transformer encoder**, which includes:

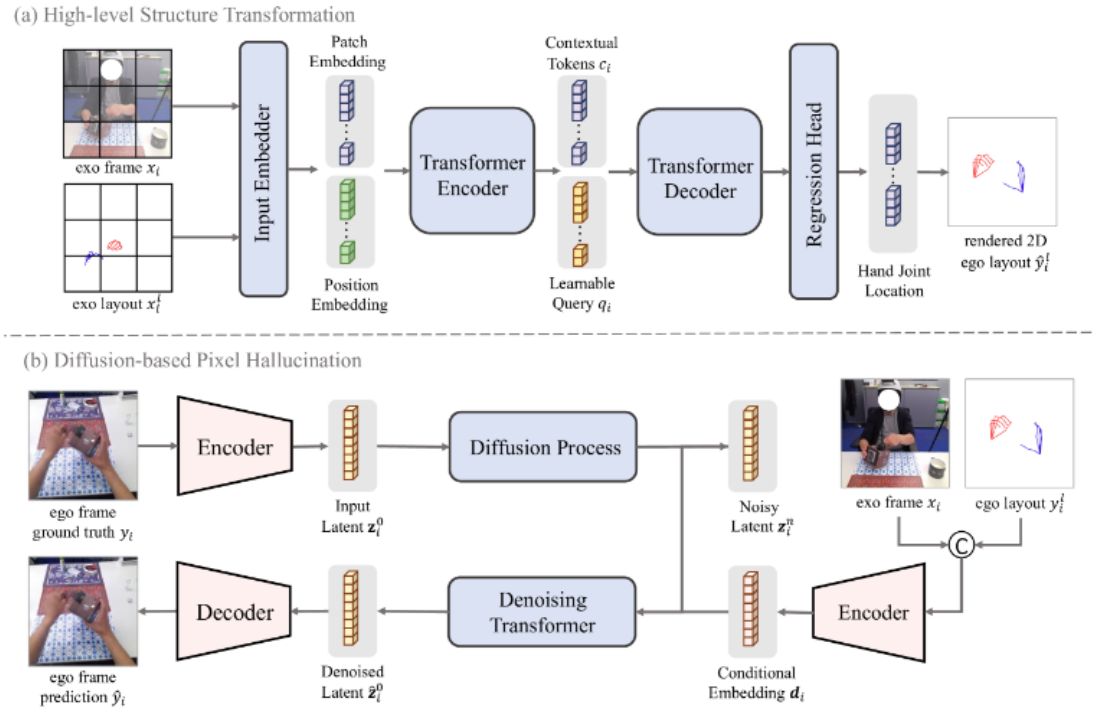

Figure 2.4: Exo2Ego Framework

- **Multi-head Self-Attention:** This allows the tokens to communicate with one another, enabling the model to capture dependencies across the entire exocentric scene.
- **Normalization and Feedforward Networks (MLP):** These operations are standard in transformer layers, ensuring efficient feature extraction and interaction between tokens.

The output of the transformer encoder is a set of **contextual tokens** $C_t^x \in R^{L\times D}$, which encapsulates the relevant high-level features of hand-object interactions in the exocentric view.

Layout Decoder: The contextual tokens C_t^x are combined with a sequence of learnable query embeddings $Q_t^e \in R^{L\times D}$, which act as a template for generating the egocentric layout is given as input.The combined embeddings are passed through L' decoder layers, which, like the encoder layers, use multi-head self-attention and normalization. However, here the focus is on generating an output sequence that corresponds to the layout in the egocentric view. The output of the decoder represents the hand positions in the egocentric layout. These are predicted as 2D joint coordinates restricted to the range $[0, 1]$, which represent the relative positions of hand joints in the egocentric view. The predicted hand joints $\hat{y}_i$ are optimized against the ground-truth joints y_i using a **bipartite matching loss** :

$$\mathcal{L}_{\text{match}} = \sum_i \|\hat{y}_i - y_i\|_2^2$$

The result is the hand-object interaction layout for the egocentric view L_t^e, which forms the structural basis for the next stage—pixel-level synthesis.

Diffusion-based Pixel Hallucination

This component refines the rough egocentric layout generated in the High-level Structure Transformation and produces photorealistic ego frames using a **conditional diffusion model**.

Latent Space Encoding : A pre-trained Variational Auto Encoder (VAE) is used to encode the ground-truth ego frame Y_t and the conditional inputs—exo frame X_t and predicted ego layout L_t^e—into a latent space. This provides a compressed representation that captures essential features for the generation

process. The encoded inputs are transformed into latent vectors, denoted as z_t for the ground-truth ego frame and $\hat{z}_t$ for the predicted latent vector.

Conditional Diffusion Model : The diffusion model starts with a noise vector sampled from a Gaussian distribution and progressively denoises it, conditioned on the latent vectors derived from the VAE. The goal is to iteratively generate a realistic ego frame based on the predicted layout. During training, a noise schedule gradually adds noise to the latent vector z_t at each step. The diffusion model is trained to reverse this process, removing noise step by step during inference. A squared error loss measures the difference between the predicted latent vector $\hat{z}_t$ and the ground-truth latent vector z_t:

$$\mathcal{L}_{\text{diff}} = \|\hat{z}_t - z_t\|_2^2$$

This loss ensures that the generated ego frames closely match the real ones.

The diffusion model generates ego frames on a per-frame basis, meaning that each frame is generated independently without relying on the previously generated frames.

2.4.2 Key Highlights of Exo2Ego Framework

- **Two-Stage Approach:** The Exo2Ego framework uses a well-thought-out two-stage architecture to translate exocentric views into egocentric perspectives.
- **Transformer-based Layout Prediction:** The architecture uses a transformer to model the spatial correspondence between exocentric and egocentric views. Transformers effectively capture long-range dependencies, which are crucial for understanding hand positioning from different perspectives, ensuring accurate spatial layout inference of hand-object interactions.
- **Diffusion-based Image Synthesis:** The diffusion process generates high-quality images, excelling at producing complex textures and fine details, resulting in realistic egocentric frames from the layout and exocentric context.
- **Handling Hand-Object Interactions:** The model emphasizes hand-object interactions, capturing the critical elements of egocentric video translation.

2.4.3 Limitations of Exo2Ego Framework

- **Assumption of Consistent Camera Angles:** The model assumes consistency in camera angles and positions between exocentric and egocentric views, but real-world scenarios with varying camera placements, user movements, or environmental changes may challenge this assumption, limiting generalization in dynamic situations.
- **Limited Generalization to Diverse Environments:** While the method works well in controlled environments with specific hand-object interactions, generalization to diverse or cluttered scenes could be limited.
- **Reliance on Pre-trained Models:** While pre-trained VAE encoders help leverage learned features, they could introduce bottlenecks.
- **Diffusion Process Efficiency:** Diffusion models, though powerful, are computationally expensive and slow to converge, making them less ideal for real-time applications like virtual or augmented reality, where fast frame generation is essential.
- **Focus on Hand-Object Interactions:** The model performs best with hand-object interactions but struggles in scenes requiring reasoning about other body parts or objects beyond the immediate hand region, limiting its application to tasks dominated by hand actions.
- **Training Data Dependency:** The model's performance is sensitive to the quality and the amount of training data. If the data is limited to a narrow set of interactions or perspectives, the model may struggle to generalize to unseen actions or objects, leading to poor performance in out-of-distribution scenarios.

2.5 Gaps in Exsisting methods for Exo-to-Ego View Translation

Exo-to-ego view translation is challenging due to drastic viewpoint changes, occlusions, and sparse input data. While methods like Scene Representation Networks (SRN), Neural Radiance Fields (NeRF) have advanced novel view synthesis, their applications in exo-to-ego translation are limited by certain characteristics.

One common issue across these models is handling occlusions. Since exocentric views often miss parts of the scene, like hands or objects, translating them to egocentric views requires precise inference about the occluded regions. Geometry-based methods like NeRF and SRN struggle with predicting occluded elements consistently. Even generative approaches like Parallel GAN and Exo2Ego encounter difficulties with ambiguity in occluded areas, making their outputs less reliable.

Large viewpoint changes present another challenge. NeRF and SRN perform well for tasks involving small view changes but fail to generalize when significant perspective shifts are needed for exo-to-ego translation. While Parallel GAN and Exo2Ego are more tolerant of larger transformations, they require extensive training to generalize and still show inaccuracies with large viewpoint shifts.

Complex models like SRN and NeRF also rely on dense input data and precise camera parameters, which are rarely available in real-world scenarios. Although Parallel GAN and Exo2Ego relax these requirements, their performance still degrades when input data is sparse, especially when scene information is incomplete.

Lastly, generative models like Parallel GAN and Exo2Ego can produce plausible results with missing data, but they are less precise compared to geometry-based methods. This makes them more prone to errors in complex exo-to-ego translations, particularly when large extrapolations are required. Overall, existing approaches face significant challenges with occlusions, large viewpoint changes, and sparse inputs, limiting their practical application in exo-to-ego tasks.

2.6 Summary

This project addresses some of these gaps mentioned in the above section by using Gaussian splatting for exo-to-ego view translation. Gaussian splatting (discussed in detail in Chapter 3) offers a more flexible and scalable approach to handling large viewpoint changes, occlusions, and sparse inputs, which are common challenges in exocentric-to-egocentric translation. By modelling the scene using Gaussian functions, a smoother and more accurate reconstruction can be achieved even when parts of the scene are occluded or missing. This method allows us to handle some of the limitations of geometry-based techniques like NeRF and SRN, as well as generative models such as Parallel GAN and the Exo2Ego framework, enhancing the robustness and precision of exo-to-ego view synthesis.

CHAPTER 3

Methodology

3.1 Overview

The method for translating exo-to-ego views starts with an exocentric scene and uses COLMAP [13] to extract sparse point clouds, which represent the initial structure for building the 3D scene. These sparse points initialize Gaussian splatting, a representation of the 3D scene using Gaussian primitives. Each Gaussian embeds spatial information, color, and uncertainty, which offers a flexible and efficient way of representing the scene.

A modified rendering algorithm is then used to transform this 3D scene into the desired egocentric perspective. The algorithm adjusts parameters of the Gaussian splats, such as position, covariance, and color, to project the exocentric scene as it would appear from the egocentric viewpoint. The rendering process is optimized to generate smooth, coherent, and photorealistic egocentric images.

Additionally, a probabilistic sampling algorithm enhances image selection during optimization. Unlike random sampling, it assigns higher probabilities to images closer to the egocentric view, ensuring the model focuses on keyframes most relevant to exo-to-ego transformation. This targeted sampling improves the model's efficiency and accuracy by directing optimization toward images that significantly contribute to ego-view synthesis.

This framework effectively translates exocentric views into egocentric ones by combining Gaussian splatting for 3D scene representation with a tailored training approach focused on the most relevant images, resulting in more accurate and photorealistic egocentric outputs.

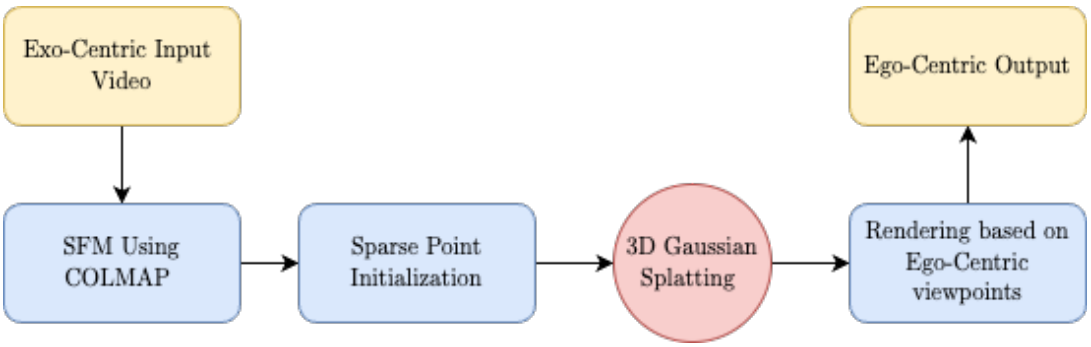

Figure 3.1: Overview of the methodology

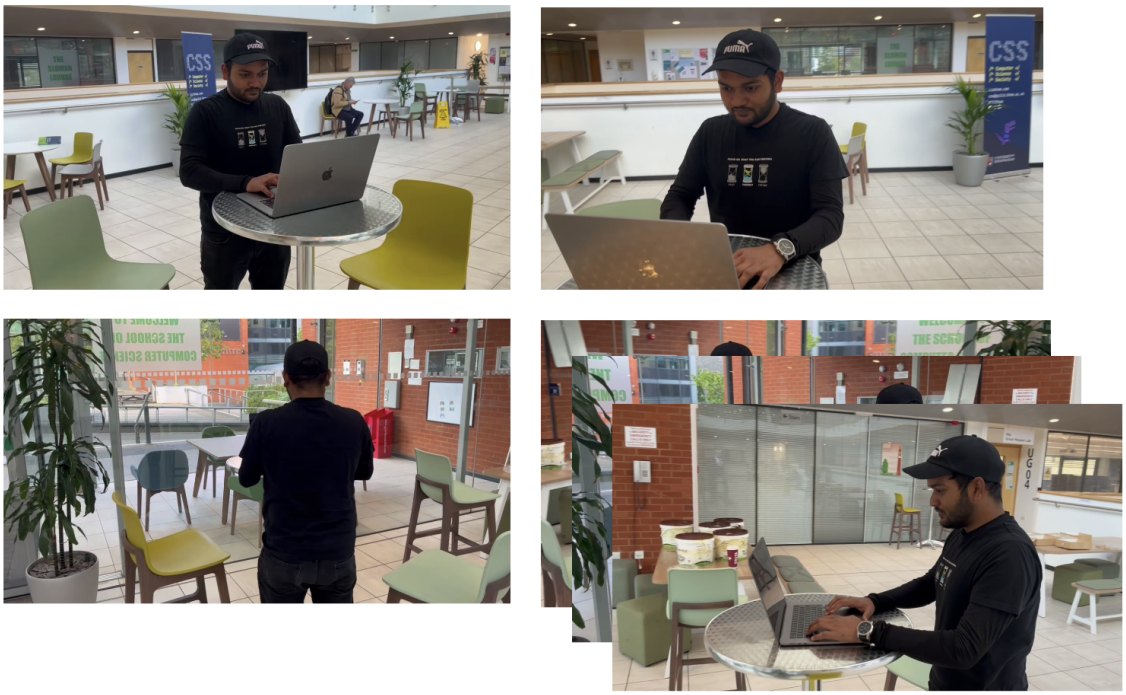

Figure 3.2: Exo-Centric Input Images

3.2 Exo-Centric Inputs

The inputs for our exo-to-ego view translation method consist of a video captured using a mobile phone, specifically designed to showcase scenes where a person is engaged in a task or job. The video is recorded in a circular motion around the individual, providing comprehensive scene coverage from various angles and perspectives. This type of recording ensures a rich dataset with diverse views, which is crucial for translating exocentric views to egocentric ones.

The video is divided into frames, typically ranging from 50 to 150 frames, depending on the video length and task complexity. These frames form the basis for the next step in the process, which involves 3D scene reconstruction and view translation. Each exocentric frame provides essential environmental information and insight into how the person interacts with objects, allowing the system to build an accurate 3D representation of the scene.

It's important to note that all the input videos and frames are custom-made and were specifically recorded for this project at the University of Birmingham using an iPhone 13. Mobile recording offers flexibility in capturing real-life scenarios without the need for specialized equipment, making the method applicable to a wide range of real-world situations. This reflects the practical conditions under which such videos are typically captured.

By using these custom inputs, the dataset is highly relevant to the intended use case of exocentric to egocentric view translation, and it allows for the development of a more robust system that can handle realistic and varied scenarios.

3.3 Structure from Motion

Structure from Motion (SfM) [13] is a technique used in computer vision and photogrammetry to reconstruct 3D structures from 2D images. The basic idea behind SfM is that the appearance of a scene varies from different viewpoints as a camera moves through space. SfM uses these changes in multiple images together to estimate the camera motion and the 3D geometry of the scene. This process is invaluable

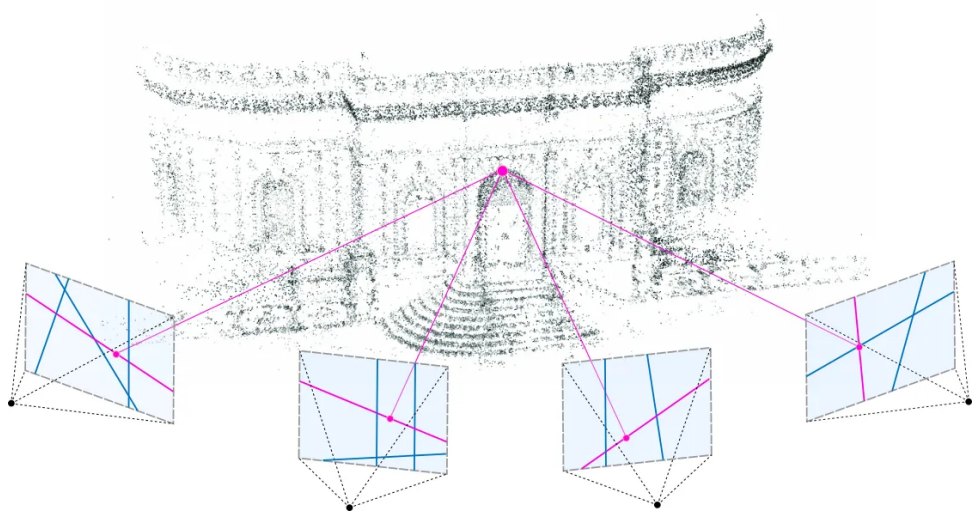

Figure 3.3: Structure From Motion

when reconstructing geometries where camera models are unknown, such as general image datasets like photographs taken from different angles without prior camera-recorded data.

SfM typically starts by capturing a sequence of 2D images, often with a camera moving around the object or scene. In our case, these images come from a video circling around a person performing a task, and these frames serve as the input for SfM. SfM works best with significant overlap between images, which allows the system to estimate both camera motion and a rough approximation of the scene structure by finding corresponding points across images. With 50 to 150 frames, a sufficient overlap for effective SfM is ensured.

Next, SfM detects distinct feature points or **keypoints** in each image. These keypoints are stable under transformations like rotation and scaling (e.g., edges, corners, textured areas). Algorithms like **SIFT** (Scale-Invariant Feature Transform) or **ORB** (Oriented FAST and Rotated BRIEF) are commonly used for this. Once detected, SfM matches these keypoints across images to identify points visible from multiple viewpoints in 3D.

After feature matching, SfM estimates the camera's pose for each image. The camera pose describes the camera's position and orientation in space. This is achieved using algorithms such as the **five-point** or **eight-point algorithms**, which utilize the geometrical relationship between matched keypoints in different images. Once the camera poses are estimated, the system can begin the 3D reconstruction process through **triangulation**. Triangulation, as shown in Fig. 3.3, uses the known camera poses and the corresponding 2D keypoints to infer the 3D coordinates of each point in space. This results in the creation of a **sparse 3D point cloud** — a collection of points representing the scene's structure.

A critical step in SfM is **bundle adjustment**, a process that optimizes both the 3D structure and camera poses to minimize errors. Bundle adjustment refines the 3D reconstruction by minimizing the reprojection error, which is the difference between where the 3D points project onto the image plane and the actual 2D keypoints detected in the images. This optimization step ensures that both the camera parameters and the 3D structure are as accurate as possible, leading to a more consistent and reliable reconstruction.

Once the sparse point cloud is built, additional processes like **Multi-View Stereo (MVS)** can be applied to generate a dense reconstruction of the scene. MVS uses multiple views of the same point to produce a dense 3D model by interpolating between the sparse points. However, for many applications, the sparse point cloud alone is sufficient as a basis for more advanced scene representation techniques, such as in our work with Gaussian splatting.

SfM offers several advantages, enabling both camera motion and 3D structure reconstruction from just 2D images, making it ideal when the camera's position is unknown or with non-sequential image sets. It provides accurate 3D reconstructions with sufficient overlap, even under varying lighting. SfM is robust, scalable, and useful in real-world applications like mapping, archaeology, and 3D models from drone footage.

However, SfM also has limitations. It requires substantial image overlap for accurate corresponding

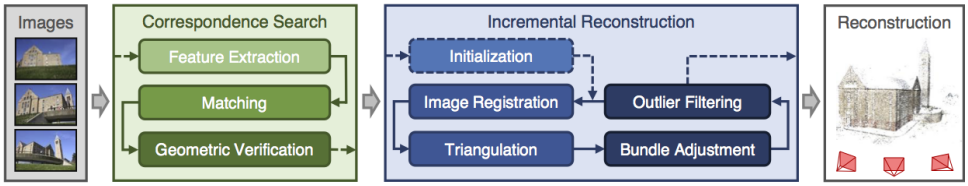

Figure 3.4: COLMAP Pipeline

points. If images have little in common or the view changes significantly, SfM may struggle to build a coherent 3D model. The sparse point cloud may not be suitable for highly detailed applications without dense reconstruction. SfM is computationally intensive, especially during the bundle adjustment step, and struggles with textureless or repetitive surfaces where distinct keypoints are hard to identify.

In our methodology, SfM is crucial for initializing 3D scene reconstruction. The sparse point cloud generated by SfM serves as a starting point for more advanced processes like Gaussian splatting. We use COLMAP for SfM, which captures the overall scene structure and camera poses, providing the foundation for advanced 3D rendering and view synthesis techniques.

3.3.1 COLMAP

COLMAP is an advanced and versatile photogrammetry software that automates the process of **Structure from Motion (SfM)** and **Multi-View Stereo (MVS)**. It was developed by Johannes Schönberger and has become a widely used tool for creating 3D models from a collection of 2D images. Its strength lies in its ability to work with unordered images, its high degree of automation, and the quality of the 3D reconstructions it produces. For our exo-to-ego view translation task, COLMAP is particularly useful for generating an accurate sparse 3D point cloud and camera poses from video frames, which can be further used for creating a detailed 3D scene representation. COLMAP performs two main tasks: feature extraction and matching to establish correspondences between images and camera pose estimation and 3D reconstruction using these matches.

For our exo-to-ego view translation, we begin by capturing a video that circles around a person performing a task. In our case, this video was recorded on an iPhone 13, though the method is flexible and can work with footage from any device, including smartphones, cameras, or even drones and other devices. The video is parsed into frames, producing a set of approximately 50 to 150 images depending on the length of the recording. These frames are then used as inputs for COLMAP to perform Structure from Motion (SfM) and reconstruct 3D scenes from 2D images.

The first step in COLMAP's pipeline involves **image preprocessing**, where the parsed frames are fed into the software. COLMAP begins by performing feature extraction, using the **SIFT** (Scale-Invariant Feature Transform) algorithm to identify distinctive keypoints in each frame. Keypoints such as corners and edges are invariant to transformations like scale and rotation, making them suitable for matching across different views of the scene. Once the features are extracted, COLMAP moves on to **feature matching**, where it identifies corresponding points between the different images. This matching process is essential, as it establishes the geometric relationships between the various perspectives captured in the video. COLMAP uses either exhaustive or guided matching techniques and applies **RANSAC (Random Sample Consensus)** to filter outlier matches, ensuring reliable correspondences.

After identifying the matches, COLMAP proceeds to camera pose estimation. It calculates the position and orientation of the camera at the time each image was captured. The software employs an **incremental Structure from Motion** approach as shown in Fig. 3.5, starting with an initial set of images and progressively adding more frames while refining the estimated camera poses and 3D points. This step is crucial for defining the camera's trajectory relative to the scene, which is essential for constructing an accurate 3D model.

With the camera poses estimated, COLMAP performs **3D point cloud reconstruction** by **triangulating** the 3D positions of the matched feature points. This results in a sparse 3D point cloud, which provides a rough geometric outline of the scene. Although this point cloud is not densely detailed, it captures the key spatial structure of the objects and the person in the scene.

To improve the accuracy of both the camera poses and the 3D points, COLMAP performs **bundle adjustment**. This optimization process reduces the reprojection error—the difference between the

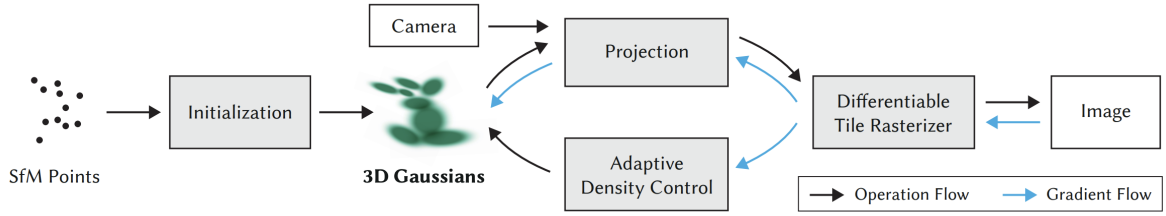

Figure 3.5: Gaussian Splatting Pipeline

observed positions of 2D points in the images and their corresponding 3D projections—thereby ensuring a more accurate and consistent 3D reconstruction. The resulting sparse 3D point cloud and precise camera pose form the basis for the next stage in our pipeline, where we use Gaussian splatting to create a detailed 3D scene representation from the initial sparse data.

In our methodology, the 3D point cloud generated by COLMAP plays a crucial role in the initialization of the Gaussian splatting process.

3.4 Gaussian Splatting

3D Gaussian Splatting [5] is a novel method for representing and rendering 3D scenes using multivariate Gaussians. It replaces traditional point clouds or neural network-based models like Neural Radiance Fields (NeRF). Unlike NeRF, which relies on volumetric ray-marching and neural networks to model and synthesize novel views, 3D Gaussian Splatting directly models the scene using a discrete set of 3D Gaussians. This approach offers the advantages of an explicit scene representation while maintaining differentiability, allowing for efficient optimization and real-time rendering.

In the 3D Gaussian splatting method, the Gaussians are defined by a full 3D covariance matrix Σ in world space, centered at a point $\boldsymbol{\mu}$. The mathematical formulation of a 3D Gaussian is given by:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

This Gaussian is multiplied by an opacity factor α during the blending process. However, to render these 3D Gaussians, it needs to be projected into a 2D space. According to the work of Zwicker et al. (2001), this projection is achieved by transforming the covariance matrix Σ into camera coordinates using a viewing transformation matrix W. The resulting covariance matrix Σ' in camera coordinates is given by:

$$\Sigma' = JW\Sigma W^\top J^\top$$

where J is the Jacobian of the affine approximation of the projective transformation. After projection, skipping the third row and column of Σ' produces a 2x2 variance matrix, which retains the same properties as if starting from planar points with normals. This allows efficient rendering using projected 2D splats.

An important consideration in Gaussian splatting is the optimization of the covariance matrix Σ. A naïve approach would involve directly optimizing Σ to represent the radiance field. However, covariance matrices must remain **positive semi-definite** to have physical meaning. Standard gradient descent can produce invalid covariance matrices during updates, which can be problematic.

To address this, the covariance matrix Σ is represented in a more intuitive form that facilitates optimization. Specifically, Σ is analogous to describing an ellipsoid, which can be broken down into two components: **a scaling matrix** S and a **rotation matrix** R. The relationship between these components is:

$$\Sigma = RSS^\top R^\top$$

To allow independent optimization of both factors, they are stored separately: a 3D vector $\mathbf{s}$ for scaling and a quaternion q for rotation. The quaternion q is normalized to maintain valid rotations, ensuring that Σ remains valid. These parameters can then be combined to form the covariance matrix during the optimization process.

In order to avoid significant computational overhead caused by automatic differentiation during training, the gradients for all parameters are explicitly derived. This enables the training process to optimize the 3D Gaussian properties efficiently. The optimization of anisotropic covariance matrices allows the Gaussians to adapt to the varying geometry in different scenes, resulting in a compact and accurate scene representation.

This representation is particularly beneficial in ensuring both the efficiency and accuracy of the Gaussian splatting process, enabling high-quality novel view synthesis and fast rendering.

3.4.1 Optimization of 3D Gaussians for Free-View Synthesis

The 3D Gaussian splatting optimization approach iteratively refines a dense set of 3D Gaussians to represent a scene accurately. In this process, not only are the positions (p), opacity (α), and covariance matrices (Σ) optimized, but the Spherical Harmonic (SH) coefficients that represent color (c) are also fine-tuned to capture the view-dependent appearance of the scene. This enables the method to handle both geometric details and view-dependent lighting effects, producing high-quality renderings from novel viewpoints.

The **optimization process** involves a series of rendering iterations. The rendered output is compared against the ground truth images from the captured dataset. This comparison drives the adjustments of various parameters. The optimization process relies on **Stochastic Gradient Descent (SGD)**, which iteratively adjusts the parameters of the 3D Gaussians to minimize the loss function. By using GPU-accelerated frameworks and CUDA kernels, the optimization can be performed efficiently. The efficiency of the optimization is further enhanced by the fast rasterization step, a critical computational bottleneck optimized for real-time performance.

The optimization also considers the **covariance matrix (Σ)** of the Gaussians, which plays a crucial role in representing large homogeneous areas of the scene with a compact set of Gaussians. To initialize the covariance matrix, an isotropic Gaussian is assumed, where the axis lengths are set equal to the mean distance to the three closest neighbouring points. This provides a reasonable starting point for the optimization.

To refine the covariance matrix during training, an **exponential activation function** is used, which ensures smooth gradients and stability during optimization. This choice helps prevent the covariance from becoming too large or too small, maintaining the anisotropy (directional stretching or compressing) required to accurately capture complex scene geometry.

For the **opacity parameter** α, a sigmoid activation function is applied to constrain the values within the range $[0, 1)$, ensuring that opacity values are smoothly optimized. This smoothness is critical for achieving high-quality blending during the rendering process.

The optimization also incorporates an exponential decay scheduling technique, similar to the approach used in Plenoxels by Fridovich-Keil and Yu (2022), which is applied to the positions of the 3D Gaussians. This scheduling gradually reduces the learning rate for position updates, allowing finer adjustments as the optimization progresses.

The **loss function** that guides the optimization is a combination of L1 loss and a D-SSIM (Differentiable Structural Similarity) term. The loss function is formulated as:

$$L = (1 - \lambda)L1 + \lambda L_{\text{D-SSIM}}$$

where $L1$ is the traditional pixel-wise difference between the rendered output and the ground truth, and $L_{\text{D-SSIM}}$ measures the structural similarity between the images, focusing on perceptual differences rather than pixel differences. The parameter λ controls the balance between the two loss terms, allowing the optimization to focus on both fine-grained pixel accuracy (via $L1$) and overall structural fidelity (via D-SSIM). This combination ensures that the resulting 3D Gaussian representation is accurate in terms of pixel-level color reproduction and maintains the scene's structural consistency.

3.4.2 Adaptive Densification in 3D Gaussian Splatting

Adaptive densification is a key component of the 3D Gaussian splatting technique. This makes the process to dynamically refine and control the distribution of 3D Gaussians in a scene. It starts from a sparse set of points obtained through **Structure from Motion (SfM)**, and the system progressively increases the density of the Gaussians to better represent the scene's geometry and appearance. This

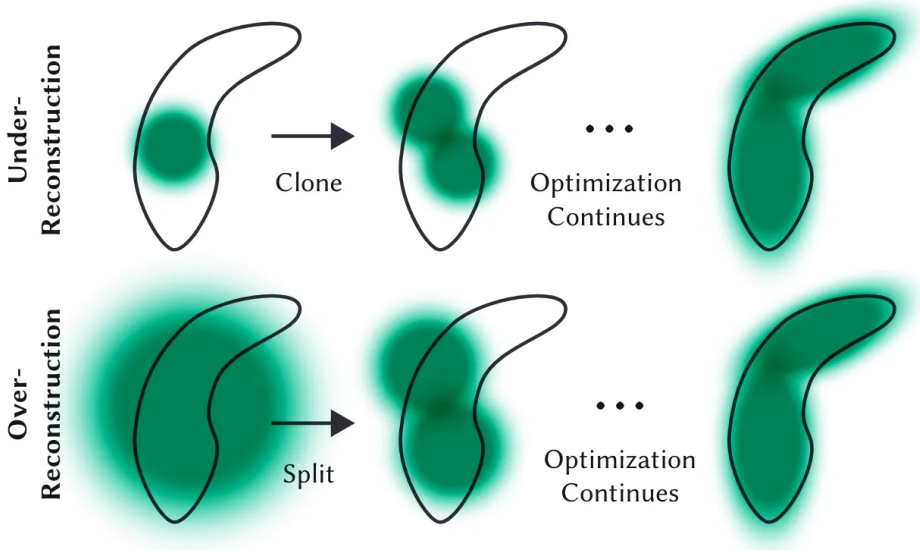

Figure 3.6: Adaptive Densification

approach ensures that both **under-reconstructed** and **over-reconstructed** regions are effectively handled. This enables the system to generate a more accurate and efficient representation of the scene for novel view synthesis.

The initial set of Gaussians is derived from the sparse point cloud obtained via SfM. Since this point cloud is often insufficient for high-quality rendering, the densification process is triggered after an **optimization warm-up** phase. This involves rendering the initial sparse Gaussians and adjusting their parameters through iterative optimization. Once the warm-up is complete, the system begins to **densify** the scene every 100 iterations by adding new Gaussians to empty regions or splitting existing ones where necessary. At the same time, Gaussians with opacity (α) values below a threshold (ϵ_α) are culled, removing Gaussians that do not significantly contribute to the final scene representation.

The **adaptive control** mechanism focuses on regions that exhibit either **under-reconstruction** (where geometric features are missing) or **over-reconstruction** (where Gaussians are too large or overlap excessively). Both of these conditions can be detected by analyzing the **view-space positional gradients** of the Gaussians. Large positional gradients often indicate areas where the geometry has not yet been fully captured or where there is excessive geometric coverage, which leads to poor rendering quality.

Under-Reconstruction: In areas where the scene is under-reconstructed, the Gaussians are typically small and spaced too far apart. These areas require the addition of new Gaussians to fill in the missing geometry. To do this, the system **clones** existing Gaussians, creating duplicates of the same size and moving them in the direction of the positional gradient. This process effectively increases the density of Gaussians in regions that need additional geometric representation.

Over-Reconstruction: Conversely, in areas where the Gaussians are too large and cover too much of the scene (over-reconstruction), the system splits these Gaussians into **smaller Gaussians**. This is done by dividing their scale by a factor of $\phi = 1.6$, which was determined experimentally to provide the optimal balance between coverage and computational efficiency. The new Gaussians are positioned by sampling the original 3D Gaussian as a **Probability Density Function (PDF)**, ensuring that the new Gaussians are appropriately distributed within the scene.

By addressing both under- and over-reconstruction in this way, the system can dynamically adjust the density and size of the Gaussians to more accurately capture the geometry and appearance of the scene.

During densification, the **total volume** of Gaussians within the scene is carefully controlled. In under-reconstructed regions, the system increases both the total volume and the number of Gaussians by adding new geometry. In over-reconstructed regions, the system splits Gaussians but conserves the total volume while increasing the number of Gaussians. This dual strategy ensures that the Gaussians are distributed optimally, representing both large homogeneous regions and fine geometric details efficiently.

A potential issue during optimization is the emergence of **floaters**, or Gaussians that become densely packed near the input camera positions. This can lead to an unjustified increase in the density of Gaussians, which may negatively impact rendering performance and accuracy. To prevent this, the system periodically resets the opacity (α) values of all Gaussians to a near-zero value every 3000 iterations. This forces the system to re-evaluate which Gaussians are actually necessary, allowing it to remove redundant Gaussians with α values below the threshold ϵ_α.

Additionally, Gaussians that become excessively large in world space or have a disproportionately large footprint in view space are periodically culled. This prevents large, overlapping Gaussians from dominating the scene, ensuring that the scene remains compact and well-represented by a manageable number of Gaussians.

The **adaptive densification** approach ensures that the set of 3D Gaussians evolves dynamically throughout the optimization process, allowing for the creation of a dense and accurate scene representation starting from a sparse set of points. By detecting under- and over-reconstructed regions, cloning or splitting Gaussians as needed, and culling redundant ones, the system maintains a **compact and efficient representation** of the scene. This, combined with the continuous adjustment of Gaussian opacity, scale, and position, allows for high-quality rendering of novel views with minimal computational overhead.

3.4.3 Fast Differentiable Rasterizer for 3D Gaussian Splatting

In 3D Gaussian splatting, efficient rendering and gradient computation are crucial for the system to handle large, complex scenes. To achieve fast rendering and optimization, a **fast differentiable rasterizer** for 3D Gaussian splats is implemented. This rasterizer is optimized for both forward and backward passes, enabling fast computation of gradients without excessive memory overhead.

The primary innovation in the rasterization process is the **tile-based approach**, which divides the screen into 16x16 tiles. This method reduces the complexity of managing individual pixels and allows efficient parallel processing. Each Gaussian is culled against the view frustum and the tiles to ensure that only relevant Gaussians are processed.

1. **Culling Against the View Frustum:** We cull 3D Gaussians by checking whether their 99% confidence interval intersects with the view frustum. This ensures that only Gaussians that can affect the rendered image are processed. Gaussians that lie too close to the near plane or outside the frustum are trivially rejected.

2. **Tile-Based Gaussian Assignment:** Each Gaussian is instantiated according to the number of tiles it overlaps. For each instance, we generate a key that combines the view-space depth and the tile ID. This allows efficient sorting of the Gaussians based on their depth relative to the camera, ensuring correct front-to-back ordering for rendering.

Once the Gaussians are assigned to tiles, they are sorted based on their depth using a **fast GPU Radix sort**. This sorting process is performed once for the entire image, avoiding the per-pixel sorting that has traditionally hindered performance in α-blending solutions.

By using Radix sorting, we achieve efficient depth ordering without the need for dynamic memory management or scene-specific tuning. This depth-sorted list of Gaussians per tile enables approximate α-blending, where blending errors are minimized for Gaussians whose splats are smaller than individual pixels.

Once the Gaussians are sorted, the rasterization process proceeds by launching a thread block for each tile. The threads in each block work collaboratively to load packets of Gaussians into shared memory, maximizing data locality and reducing memory transfer overhead.

For each pixel within a tile, the threads traverse the sorted list of Gaussians in front-to-back order, accumulating color and opacity (α) values. The accumulation process stops once a pixel reaches $\alpha = 1$, indicating that the pixel is fully saturated. This approach minimizes unnecessary computations and ensures that the rasterizer stops processing pixels that no longer need additional Gaussians.

The **α-blending process** in this rasterizer is approximate but highly efficient. By sorting the Gaussians before blending, the need for per-pixel sorting is avoided. This reduces the memory and computational overhead, especially in scenes with a large number of overlapping Gaussians.

For the **backpropagation step**, gradients for each Gaussian that contributed to the color and opacity of each pixel have to be computed. Unlike previous methods that store arbitrarily long lists of

blended points per pixel, this approach reuses the sorted array of Gaussians from the forward pass. This is accomplished by traversing the Gaussians in back-to-front order, ensuring that gradients are computed in the correct sequence.

To facilitate **gradient computation**, the final accumulated opacity (α) is stored at the end of the forward pass. During the backward traversal, this opacity is divided by each Gaussian's opacity, allowing the computation of required gradient coefficients without storing intermediate values. This method significantly reduces memory overhead, as only the final accumulated opacity is stored.

The fast differentiable rasterizer is a key component in making 3D Gaussian splatting efficient and scalable. By using tile-based sorting, efficient GPU Radix sort, and collaborative memory management within each thread block, this rasterizer enables real-time rendering of complex scenes while maintaining differentiability. The approximate α-blending, combined with efficient gradient computation, ensures that the system can handle arbitrary scene complexity without hard limits on the number of splats receiving gradients. This approach is crucial for enabling the dynamic optimization of Gaussian parameters (position, covariance, color, etc.) and maintaining high-quality rendering performance.

3.5 Optimized Training Algorithm for Ego-Centric View Rendering

In the standard Gaussian splatting training algorithm, the training process typically involves random sampling of input images followed by optimization of the Gaussians based on the sampled views. However, in the optimized approach for ego-centric view rendering, a probabilistic sampling strategy that assigns higher probabilities to images closer to ego-centric viewpoints is employed. This ensures that views that are more relevant to the user, such as those from the ego-centric perspective, are prioritized during training. This makes the system become more efficient and effective in refining Gaussian representations, particularly for the most critical viewpoints.

The key concept in this approach is to calculate a distance between the training cameras and the hypothetical camera parameters from an ego-centric perspective, which serves as a measure of their relevance. This distance is determined based on several factors, including the spatial translation, rotational differences, and variations in the cameras' field of view (FOV). Specifically, the distance between two cameras is calculated as a combination of their translation distance, rotation distance (expressed through quaternion representations), and differences in their horizontal and vertical FOVs. The translation distance captures the Euclidean distance between the camera positions, while the rotational distance is computed using quaternions, which offer an efficient way to quantify angular differences between two camera orientations. The FOV distances represent how much the cameras differ in their viewing angles, both horizontally and vertically. We assign weights to these factors, allowing for flexible tuning of the importance of each parameter during the calculation.

Once the distance between each training camera and ego-centric viewpoints is computed, we proceed to assign probabilities to each training camera based on its proximity to the ego-centric viewpoint. Cameras that are closer to ego-centric viewpoints are considered more relevant and thus receive higher probabilities for selection during training. The distances are normalized using the maximum distance in the set, effectively inverting the relationship so that smaller distances (closer cameras) translate into higher probabilities. A small constant is added to avoid numerical instability, ensuring that the system remains robust during optimization.

In summary, this optimized training algorithm enhances the Gaussian splatting process by focusing more on views that are relevant to ego-centric perspectives. The probabilistic sampling mechanism, driven by a well-defined camera distance metric, allows the system to converge faster and produce higher-quality results for free-view rendering, particularly when the ego-centric viewpoint is prioritized. This targeted approach ensures better scene understanding and refinement for critical viewpoints, improving the overall rendering performance.

3.6 Redering Based on Ego-Centric View Points

In this approach to rendering based on ego-centric viewpoints, the primary inputs for the rendering algorithm are the camera extrinsic parameters of the hypothetical ego-centric viewpoint. These extrinsic parameters include the camera's rotation matrix R and translation vector T, which together define the

position and orientation of the camera in the world coordinate system. Using these parameters, the algorithm generates images from the perspective of an observer situated within the scene, simulating a first-person, immersive view.

The process begins with the determination of the ego-centric camera parameters, which are derived from test images using COLMAP or similar Structure-from-Motion (SfM) tools. COLMAP computes the extrinsic parameters R and T by aligning the camera's view to the scene's 3D geometry, representing how an observer would see the scene from a given viewpoint.

Once the camera extrinsics are obtained, they are fed into the rendering algorithm. The rendering algorithm works by projecting the 3D scene, represented by a collection of Gaussians or 3D primitives, onto the 2D image plane of the hypothetical ego-centric camera. By focusing only on the extrinsic parameters, our approach allows for flexibility in rendering. Any hypothetical viewpoint can be simulated by providing new extrinsic parameters without retraining or reprocessing the entire scene.

The core benefit of this approach is that the scene representation remains constant while only the camera's position and orientation change. This makes rendering fast from any ego-centric viewpoint, thereby creating realistic visual outputs tailored to an observer's specific perspective on the scene. This method is especially powerful for immersive applications, as it directly reflects the user's viewpoint without the need for complex re-computations.

3.7 Evaluation Metrics

In this project, the evaluation metrics used to assess the quality of generated images in ego-centric view rendering and view translation tasks include **Structural Similarity Index (SSIM)**, **Peak Signal-to-Noise Ratio (PSNR)**, and **Learned Perceptual Image Patch Similarity (LPIPS)**. Each metric measures different aspects of image quality, ranging from structural fidelity to perceptual similarity. Below is a detailed explanation of these metrics:

3.7.1 Structural Similarity Index (SSIM)

The Structural Similarity Index (SSIM) is a perceptual metric that quantifies image quality by measuring the similarity between two images. SSIM is based on the idea that the human visual system is highly sensitive to changes in image structure, brightness, and contrast. Unlike pixel-wise comparisons, SSIM considers changes in structural information, which makes it more aligned with human perception.

SSIM Formula

The SSIM between two images x and y is calculated using the following equation:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where:

- μ_x and μ_y are the means of images x and y
- σ_x^2 and σ_y^2 are the variances of x and y
- σ_{xy} is the covariance between x and y
- C_1 and C_2 are small constants to stabilize the division.

Interpretation

- SSIM values range from -1 to 1, with 1 indicating perfect structural similarity and -1 indicating complete dissimilarity.
- SSIM focuses on perceptual quality and is sensitive to structural distortions like blurring or compression artifacts.
- SSIM is a widely used metric for tasks like image generation, image restoration, and view synthesis due to its ability to capture both local and global image structures.

3.7.2 Peak Signal-to-Noise Ratio (PSNR)

The Peak Signal-to-Noise Ratio (PSNR) is a widely used metric for measuring the quality of reconstructed images by comparing the original and generated images pixel by pixel. PSNR quantifies how much noise or distortion is present in the generated image relative to the original. The higher the PSNR, the better the quality of the generated image.

PSNR Formula

The PSNR is defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

Where:

- MAX_I is the maximum possible pixel value of the image (e.g., 255 for 8-bit images).
- MSE is the Mean Squared Error between the original image x and the generated image y, calculated as:

$$MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (x(i,j) - y(i,j))^2$$

Here, m and n are the dimensions of the images, and $x(i,j)$ and $y(i,j)$ are the pixel values at location (i,j) in the original and generated images, respectively.

Interpretation

- PSNR values are typically expressed in decibels (dB), and higher values indicate better quality (less noise or distortion).
- PSNR is sensitive to slight variations in pixel values, making it less suitable for evaluating perceptual quality in some cases.

3.7.3 Learned Perceptual Image Patch Similarity (LPIPS)

The Learned Perceptual Image Patch Similarity (LPIPS) is a perceptual similarity metric that measures the difference between two images based on features extracted by a pre-trained neural network. Unlike SSIM and PSNR, which rely on pixel-wise differences, LPIPS compares the perceptual features at different layers of a deep neural network, making it better at capturing perceptual differences that align with human vision.

LPIPS Calculation:

LPIPS is computed by passing the original and generated images through a pre-trained deep convolutional neural network (e.g., VGG or AlexNet) and comparing their feature representations at various layers. The formula for LPIPS is:

$$LPIPS(x,y) = \sum_{l} \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \| f_l(x)_{h,w} - f_l(y)_{h,w} \|^2$$

Where:

- $f_l(x)$ and $f_l(y)$ are the feature maps at layer l of the neural network for images x and y, respectively.
- H_l and W_l are the height and width of the feature maps at layer l.

Interpretation

- LPIPS values range from 0 to 1, where 0 indicates identical perceptual quality and 1 indicates significant perceptual differences.
- Lower LPIPS scores indicate that the generated image is perceptually more similar to the original.
- LPIPS is particularly useful in tasks like image generation, where perceptual quality is more important than exact pixel-wise matching.

Together, these metrics provide a comprehensive evaluation of both pixel-level accuracy and perceptual similarity, which is crucial for the quality assessment of rendered ego-centric views and 3D view translations.

3.8 Summary

This chapter presented the methodology behind our approach to ego-centric view rendering using **3D Gaussian splatting**. The primary idea is to represent a scene using 3D Gaussians, each characterized by its position, color, and covariance matrix. These Gaussians are optimized using **stochastic gradient descent** to capture view-dependent appearance, with spherical harmonic coefficients used to model the color variations.

A key part of this process is **adaptive densification**, which refines the initial sparse point cloud from Structure-from-Motion (SfM) by adding or splitting Gaussians in under- or over-reconstructed areas. This ensures a more accurate representation of the scene. Additionally, a **fast differentiable rasterizer** was used to handle the efficient backpropagation of gradients during training, speeding up the rendering and optimization process.

For optimizing ego-centric viewpoints, a **probabilistic sampling strategy** was implemented that prioritizes input images captured closer to ego-centric viewpoints. Camera pose distances, calculated using quaternion-based metrics, are used to assign sampling probabilities, leading to more accurate ego-centric view optimization.

In this project, the evaluation of generated images, particularly for ego-centric view rendering and view translation, is performed using three key metrics: Structural Similarity Index (SSIM), Peak Signal-to-Noise Ratio (PSNR), and Learned Perceptual Image Patch Similarity (LPIPS). Together, these metrics provide a comprehensive assessment of both the structural and perceptual quality of the rendered images, ensuring that the results are accurate in pixel detail and visually consistent with human perception.

This methodology sets the foundation for rendering high-quality ego-centric views. However, challenges such as dataset limitations, particularly for dynamic scenes, highlight areas for future work.

CHAPTER 4

Experiments and Results

4.1 View Translation Using 3d Gaussian Splatting

In this experiment, we demonstrate the process of view translation using 3D Gaussian splatting. This experiment aims to render novel ego-centric views from input videos taken from real-world scenes. We collected custom video footage using an iPhone, capturing 5 to 10 different scenes, each containing various objects, lighting conditions, and complex geometries. The process comprises several stages, from video preprocessing to generating ego-centric rendered outputs.

4.1.1 Video Preprocessing

The first step involved converting the captured videos into a sequence of frames, as shown in Fig.4.1. A custom Python function was written to extract individual frames from the video at a fixed frame rate. These frames are then used as input for COLMAP, a Structure-from-Motion (SfM) tool that reconstructs sparse 3D point clouds and estimates camera poses (intrinsic and extrinsic parameters). The Python script automates converting videos to frames, ensuring that the input frames are consistently aligned with COLMAP's pipeline.

4.1.2 COLMAP Reconstruction

Once the frames were extracted, they were fed into COLMAP to perform feature matching and camera pose estimation. COLMAP produced a sparse 3D reconstruction of the scene and the corresponding camera extrinsics for each frame, which are crucial for further processing. The output file structure from COLMAP contained:

- Sparse point cloud data representing the scene geometry.
- Camera parameters, including intrinsic and extrinsic matrices, are located in cameras.bin and images.bin files, respectively.
- Sparse depth maps of the scenes.

The sparse point cloud served as the initial 3D representation of the scene. Figure. 4.2 shows an example of COLMAP's output format. This step provides the necessary initialization for applying 3D Gaussian splatting to create a more continuous and dense scene representation.

4.1.3 3D Gaussian Splatting

Using the sparse point cloud generated by COLMAP, the next step was to perform 3D Gaussian splatting. For each point in the sparse point cloud, a Gaussian is fitted to model its radiance and spatial distribution

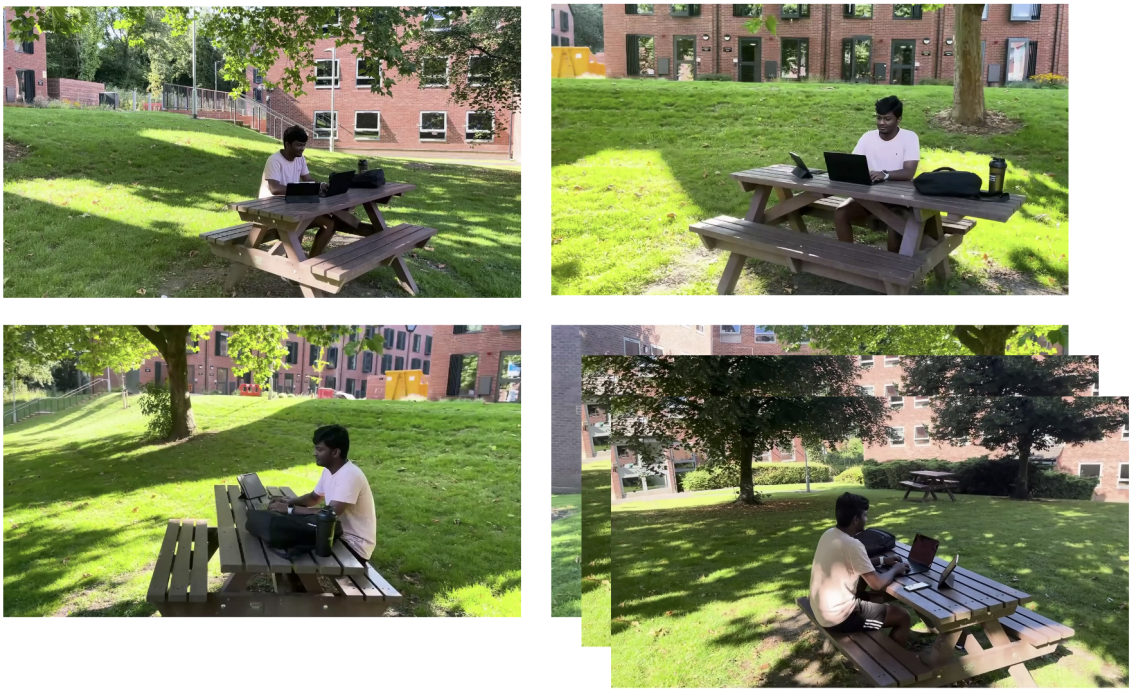

Figure 4.1: Exo-Centric Input Images

```
+── images
│   +── image1.jpg
│   +── image2.jpg
│   +── ...
+── sparse
│   +── 0
│   │   +── cameras.bin
│   │   +── images.bin
│   │   +── points3D.bin
│   +── ...
+── dense
│   +── 0
│   │   +── images
│   │   +── sparse
│   │   +── stereo
│   │   +── fused.ply
│   │   +── meshed-poisson.ply
│   │   +── meshed-delaunay.ply
│   +── ...
+── database.db
```

Figure 4.2: Colmap Output Format

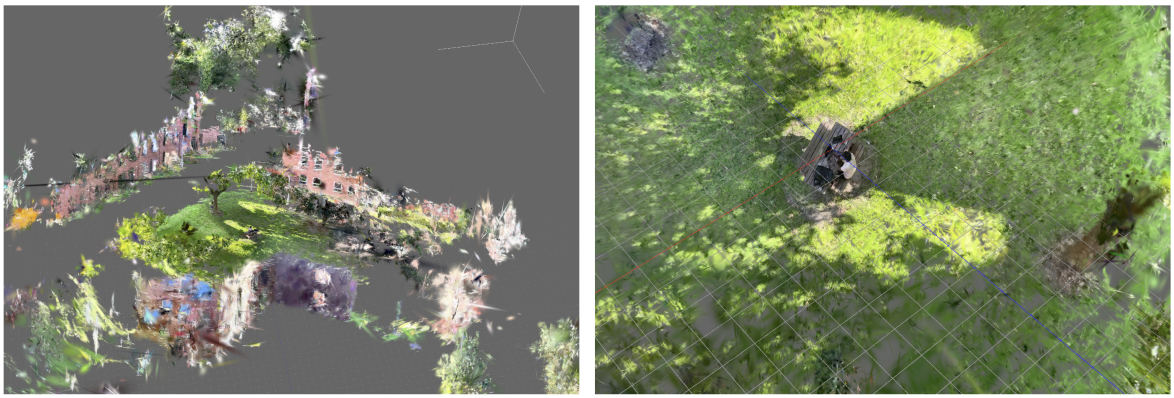

Figure 4.3: Scene Rendered in 3D using Gaussian Splatting

Figure 4.4: Scene Rendered in 3D using Gaussian Splatting

in the scene. Each 3D Gaussian is parameterized by its mean (location), covariance (shape), and opacity. This representation allows for a smooth, continuous radiance field without needing a neural network-based model, like in NeRFs.

The Gaussian splatting process was optimized over several iterations, refining the position, opacity, and covariance of the Gaussians to best represent the scene. Fig. 4.3 and Fig. 4.4 show a scene of a person working rendered in 3D using Gaussian Splatting. The splatted Gaussians provide a realistic scene reconstruction with anisotropic shapes that capture fine details and accurately reflect lighting variations.

4.1.4 Ego-Centric View Rendering

Finally, using the camera extrinsics from COLMAP, novel ego-centric views of the scene were rendered. By inputting camera extrinsics corresponding to various ego-centric viewpoints, it can generate rendered outputs as if viewed from a first-person perspective. This simulates how the scene would appear if observed from different positions within the 3D environment.

The rendering algorithm was modified to accept these camera extrinsic parameters and project the Gaussian splats onto a 2D image plane, generating a realistic visual output from the perspective of the ego-centric camera. The outputs captured detailed spatial structures, maintaining a smooth transition between viewpoints. Fig. 4.5 showcases several examples of rendered ego-centric images, showing how the scene changes as the viewpoint shifts.

Figure 4.5: Rendered Ego-Centric Outputs

4.1.5 Evaluation Metrics and Comparison

The experiments focused on translating these exocentric views into egocentric ones using the 3D Gaussian splatting method, and the results were evaluated using three key metrics: SSIM, PSNR, and LPIPS.

The output metrics for different scenes are summarized in Table. 4.1, where SSIM, PSNR, and LPIPS for various scenes (e.g., "Work", "CS1", "Vale1") are displayed. The metrics shown in Table. 4.1 are an average of the individual metrics of each of the ego-centric views generated. These metrics give insight into the quality of the rendered egocentric views, evaluating structural similarity, signal-to-noise ratio, and perceptual quality.

Table 4.1: Evaluation metrics for different scenes

Scene	**SSIM (↑)**	**PSNR (dB) (↑)**	**LPIPS (↓)**
work	0.50	15.21	0.42
cs1	0.70	18.83	0.33
vale1	0.80	22.62	0.27

Table 4.2 compares this method with existing baseline approaches for exo-to-ego view translation, including Pix2Pix and P-GAN. The metrics for this method represent the average of the maximum values for SSIM and PSNR obtained across all scenes in the custom dataset. This approach consistently outperforms the baselines in SSIM and has comparable PSNR values with the SOTA method, highlighting the efficacy of the Gaussian splatting technique for generating high-quality egocentric views. Although the LPIPS score for the baseline methods is not fully available, our approach achieves a satisfactory perceptual quality score, further underscoring its effectiveness.

Table 4.2: Comparison with Other Methods

Metric	**SSIM (↑)**	**PSNR (dB) (↑)**	**LPIPS (↓)**
Ours	**0.76**	23.35	0.26
Pix2Pix	0.25	15.05	-
P-GAN	0.30	17.02	-
Exo2Ego	0.16	**27.94**	-

Notably, our method could not be directly compared with state-of-the-art methods such as NeRF, SRN, or even the original Gaussian Splatting techniques. This is because, to date, no prior work has

utilized these methods specifically for exo-to-ego view translation. The metrics reported in their studies typically focus on scene reconstruction from viewpoints that are similar to the exocentric perspective and do not address the drastic viewpoint changes involved in our scenario. Consequently, the unique challenges of exo-to-ego translation explored in this work distinguish it from the existing literature, marking a novel contribution to this area of research.

Overall, the experiment demonstrates the strength of our exo-to-ego view translation approach, with consistent performance across different scenes and superior results when compared to existing methods.

4.2 View Translation Using 4d Gaussian Splatting for Dynamic Scenes

This experiment aimed to extend the Gaussian splatting technique to handle dynamic scenes using the approach presented in the paper on '4D Gaussian Splatting for Real-Time Dynamic Scene Rendering' (Wu et al., 2024) [20]. This method allows for modelling dynamic, time-varying scenes by introducing a temporal dimension (the fourth dimension) to the Gaussian splatting process, enabling more detailed and fluid scene reconstructions.

4.2.1 Objective

The goal was to implement 4D Gaussian splatting to reconstruct dynamic scenes where both the geometry and radiance of the scene vary over time. The method was first tested on the publicly available **DyNeRF** dataset and then on custom dynamic and multiview datasets such as **ExoEgo4D**, **Aria Pilot**, and **H2O**. However, challenges were encountered with custom datasets, which limited their usefulness for 4D reconstruction.

4.2.2 4D Gaussian Splatting on DyNeRF Dataset

The first successful implementation was on the **DyNeRF** dataset, which contains dynamic scenes captured from multiple viewpoints over time. The dataset includes sequences of frames, with temporal continuity across the frames, allowing 4D Gaussian splatting to model the scene's geometry and appearance at different time steps.

The implementation process started by using the same approach as traditional 3D Gaussian splatting, adding a temporal component. This method integrates 3D Gaussians with 4D neural voxels to comprehensively represent dynamic scenes. A decomposed neural voxel encoding algorithm, inspired by HexPlane, is employed to efficiently build Gaussian features from 4D neural voxels. These features are then processed by a lightweight Multi-Layer Perceptron (MLP) to predict Gaussian deformations at novel timestamps, enabling accurate scene synthesis. Each point in the scene was not only associated with a 3D Gaussian in space but also with a time component to account for dynamic changes. The 4D Gaussians were optimized iteratively, considering the changing radiance and structure of the scene over time.

The results from the DyNeRF dataset demonstrated the capability of 4D Gaussian splatting to handle dynamic scenes. The reconstructed 4D scenes were highly detailed, and the transitions between frames were smooth, preserving both spatial and temporal consistency. Fig. 4.6 shows the successful 4D reconstruction outputs from the DyNeRF dataset, capturing the detailed dynamics of objects moving through space over time. The movement of objects and the changes in lighting were well captured, indicating that this approach is effective for high-quality dynamic scene reconstruction.

4.2.3 Challenges with Custom Datasets

After successfully implementing 4D Gaussian splatting on DyNeRF, an attempt was made to extend the process to custom dynamic datasets. Experiments with monocular datasets and several multiview datasets, including **ExoEgo4D**, **Aria Pilot**, and **H2O** were conducted. However, several challenges were encountered that made it difficult to apply 4D reconstruction to these datasets.

Limited Number of Views: The multiview datasets such as ExoEgo4D, Aria Pilot, and H2O provided only 4 to 8 camera views, which was insufficient for COLMAP's initialization and 3D reconstruction. COLMAP relies heavily on a dense set of views to compute accurate camera poses and build

Figure 4.6: 4D Gaussian Splatting on DyNeRF Dataset

a reliable sparse point cloud. The limited number of views in these datasets led to poor or incomplete reconstructions, making it impossible to proceed to 4D splitting.

Technical Limitation in Capturing Multiview Dynamic Scenes: Capturing a multiview dynamic scene requires multiple cameras set up around the scene. This setup is difficult to achieve without the appropriate hardware and even more challenging when trying to ensure that each camera is accurately calibrated and synchronized. The difficulty of obtaining multiple cameras and calibrating them correctly to capture a dynamic scene was another key limitation. Without such a setup, it is not feasible to apply 4D Gaussian splatting to custom dynamic datasets.

Monocular Dataset Limitations: Monocular datasets present challenges since they lack multiple scene views. It is difficult to accurately estimate depth, geometry, and camera poses without multiple perspectives.

While we successfully implemented 4D Gaussian splatting on the DyNeRF dataset, attempts to apply the method to custom datasets were unsuccessful due to these limitations. The multiview datasets we used did not provide enough views for reliable COLMAP initialization, and monocular datasets lacked the necessary depth information for accurate 3D reconstruction. As a result, generating 4D reconstructions from these custom datasets was unsuccessful.

Despite these challenges, the experiment demonstrated that 4D Gaussian splatting is a promising approach for reconstructing dynamic scenes with a sufficiently dense and temporally continuous dataset. Future work could explore alternative methods for initializing 3D reconstructions in sparse-view scenarios or develop custom datasets with more comprehensive coverage of dynamic scenes.

4.3 Using Gaze direction for Ego-Centric View Rendering

In this experiment, the goal is to use object-aware gaze target detection as shown in Fig. 4.7 to calculate gaze direction and map it to parameters for ego-centric view rendering. The experiment utilizes the method described in the paper 'Object-aware Gaze Target Detection' [16] which focuses on object-aware gaze target detection through a Transformer-based architecture.

The object-aware gaze target detection model is designed to predict where a person is looking in an image or video by building associations between the head (gazer) and objects in the scene. This approach is particularly useful for ego-centric view rendering, as it helps identify the key objects that the user is likely focusing on and adjust the view accordingly.

In this experiment, the **object-aware gaze target detection** is employed to identify gaze direction from input exocentric frames. The system processes the frames to detect heads and other relevant objects using the Object Detection Transformer. Then, for each head, a **gaze vector** is computed, which is crucial for determining the direction in which a person is looking.

Once the gaze vector is determined, a gaze cone is constructed to model the individual's field of view. This helps in narrowing down the objects within the gaze cone, which are the most likely targets of interest. The Gaze Object Transformer refines this information to produce a heatmap that predicts the exact gaze point in the scene, which includes detecting whether the gaze is within or outside the frame.

The ultimate goal is to map the detected gaze direction to parameters for ego-centric view rendering. This involves adjusting the view to reflect what the person is looking at, potentially leading to applications in virtual reality (VR), augmented reality (AR), or head-mounted displays where the system adapts the

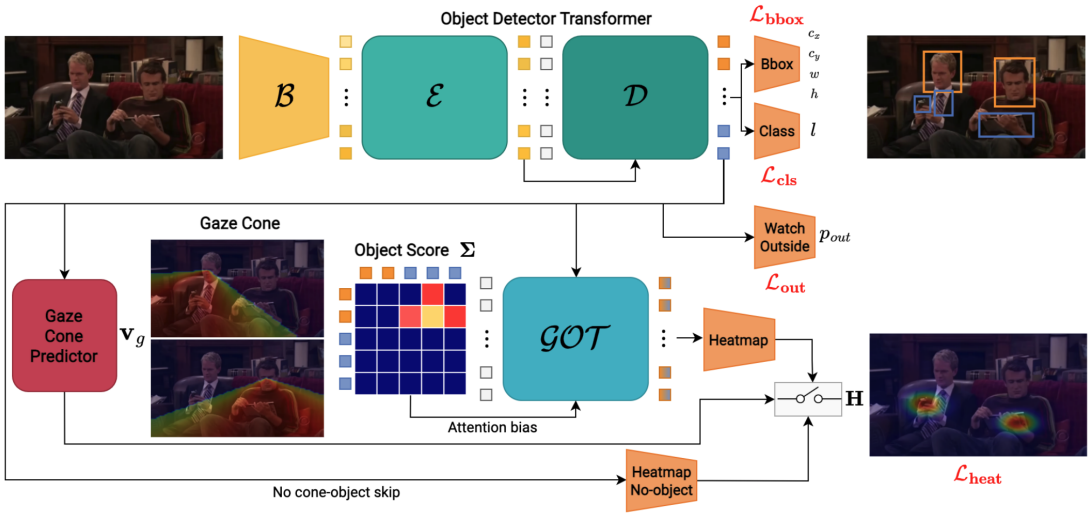

Figure 4.7: Object Aware Gaze Target Detection

view based on gaze patterns. However, this part of the work is still in progress and will be further developed in future iterations of the experiment.

4.4 Summary

In this chapter, the series of experiments that were tested to perform exo-centric to ego-centric view translation are explained. The first experiment demonstrated the ability to translate views using 3D Gaussian splatting based on custom video inputs. By converting video to frames and using COLMAP for initialization, ego-centric renderings were obtained, showcasing promising results. The second experiment explored the implementation of 4D Gaussian splatting for dynamic scenes using the Dynerf dataset. Although the approach worked well for pre-existing datasets, the limitations of capturing multiview data for custom dynamic scenes due to the need for multiple cameras with known parameters proved to be a significant challenge. The third experiment explored the use of gaze direction for ego-centric rendering, using object-aware gaze target detection. Although this process is still in progress, initial results show promising directions for integrating gaze-based input for ego-centric view generation.

Overall, these experiments underline the potential of Gaussian splatting and gaze-driven methods in the field of ego-centric rendering while also revealing the practical limitations of dataset collection, multiview scene reconstruction, and the integration of advanced gaze tracking. Future work will focus on overcoming these challenges, particularly in dynamic scene reconstruction and more accurate gaze-to-viewpoint mapping.

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

This report outlines several significant contributions in advancing the field of exocentric (exo) to egocentric (ego) view translation, with a focus on the application of 3D Gaussian splatting techniques. Through a comprehensive literature review, the existing methodologies and limitations related to exo-to-ego view translation were identified, which highlights the challenges of synthesizing high-quality ego-centric views from exo-centric viewpoints.

One of the key contributions is the application of **3D Gaussian splatting** for exo-to-ego view translation, which has not been explored before. This technique, known for its ability to efficiently represent complex 3D scenes with anisotropic Gaussian splats, was adapted to handle the challenging task of view translation. By using this approach, high-quality ego-centric views were generated from sparse 3D data, effectively bridging the gap between exocentric and egocentric perspectives. The Gaussian splatting method allowed for flexible and dense representations of scenes, and its differentiable properties made it suitable for rendering ego-centric views, an important aspect not previously addressed in this context.

To evaluate the effectiveness of the approach, a **custom dataset** containing paired exocentric and egocentric views was created. This dataset is a valuable resource for benchmarking and testing the translation methods, providing a consistent framework to evaluate the accuracy of the ego-centric outputs. It allowed to systematically test our approach in various scenes and validate the quality of the renderings.

Moreover, a **probabilistic sampling approach** was introduced during the training phase, prioritising images closer to the ego-centric viewpoints. This technique ensured that the network focused on optimizing for ego-centric perspectives, improving both the training efficiency and the final render quality. This adaptive sampling method enhances the alignment of rendered views with human-centric perspectives, making it a critical advancement in the training process.

In addition to the Gaussian splatting and probabilistic sampling, a **rendering algorithm** capable of taking in arbitrary camera extrinsics and generating corresponding rendered views was created. This algorithm enables more flexible rendering based on input camera viewpoints. It significantly enhances the ability to simulate ego-centric viewpoints from various camera configurations, making it a versatile tool for view synthesis.

Overall, the report demonstrates substantial progress in exo-to-ego view translation by leveraging advanced rendering techniques and novel training strategies. The combination of Gaussian splatting, probabilistic sampling, and a flexible rendering pipeline offers a robust framework for generating high-quality ego-centric views from exocentric data. These contributions not only address gaps in the existing literature but also lay the groundwork for future research in dynamic scene reconstruction and gaze-directed rendering for ego-centric applications.

5.2 Future Work

This report has laid the foundation for exo-to-ego view translation using new approaches like 3D Gaussian splatting. However, there are various other strategies which could significantly enhance the outcomes of this research.

Gaze Direction Mapping for Ego-Centric Rendering: One of the most promising avenues for future work is integrating gaze direction to inform the camera parameters used for rendering ego-centric views. By employing advanced gaze detection techniques such as the method outlined in the "Object-aware Gaze Target Detection" paper, the gaze direction could be accurately mapped to ego-centric camera parameters. This would allow the rendering system to adapt dynamically based on where a person looks, resulting in more personalized and realistic ego-centric viewpoints. Combining gaze-based input with camera parameter control will also improve the fidelity of human-centred renderings in virtual environments.

4D Gaussian Splatting for Dynamic Scenes: Another direction is to incorporate **4D Gaussian splatting** as a baseline for handling dynamic scenes. While this report has successfully used 3D Gaussian splatting for exo-to-ego view translation in static scenes, applying the 4D Gaussian splatting technique, particularly for video conversion, could significantly enhance the realism and temporal consistency of dynamic scene reconstructions. This approach could be tested on monocular or multiview datasets to handle complex real-world scenes, allowing for more effective video-based ego-centric rendering.

Utilizing MVSplat or PixelSplat for Minimal Image Inputs: A major limitation identified in our work was the requirement for a significant number of input images to generate high-quality ego-centric views. **MVSplat** [3] or **PixelSplat** [2], which were introduced very recently, are designed to work with a very low number of input images (as few as 2) and present a promising solution to this issue. Future work could focus on integrating these approaches to achieve 3D reconstructions and ego-centric view generation from minimal data. This would open up new possibilities for ego-view synthesis in scenarios with limited visual input, significantly broadening the applicability of the technique to diverse datasets.

By pursuing these directions, future research can further enhance the accuracy, flexibility, and practical usability of exo-to-ego view translation.

Bibliography

[1] E. R. Chan, K. Nagano, M. A. Chan, A. W. Bergman, J. J. Park, A. Levy, M. Aittala, S. D. Mello, T. Karras, and G. Wetzstein. Generative novel view synthesis with 3d-aware diffusion models. *arXiv preprint arXiv:2304.02602*, 2023.

[2] D. Charatan, S. Li, A. Tagliasacchi, and V. Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction, 2024.

[3] Y. Chen, H. Xu, C. Zheng, B. Zhuang, M. Pollefeys, A. Geiger, T.-J. Cham, and J. Cai. Mvsplat: Efficient 3d gaussian splatting from sparse multi-view images, 2024.

[4] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[5] B. Kerbl, C. Reiser, M. Niessner, J. Thies, and T. Müller. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (TOG)*, 42(4):1–10, 2023.

[6] G. Liu, H. Tang, H. Latapie, and Y. Yan. Exocentric to egocentric image generation via parallel generative adversarial network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1843–1847. IEEE, 2020.

[7] G. Liu, H. Tang, H. M. Latapie, J. J. Corso, and Y. Yan. Cross-view exocentric to egocentric video synthesis. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 974–982, 2021.

[8] M. Luo, Z. Xue, A. Dimakis, and K. Grauman. Put myself in your shoes: Lifting the egocentric perspective from exocentric videos. *arXiv preprint arXiv:2403.06351*, 2024.

[9] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[10] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. M. Sajjadi, A. Geiger, and N. Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5480–5490, 2022.

[11] X. Ren and X. Wang. Look outside the room: Synthesizing a consistent long-term 3d scene video from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3563–3573, 2022.

[12] R. Rombach, P. Esser, and B. Ommer. Geometry-free view synthesis: Transformers and no 3d priors. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14356–14366, 2021.

[13] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016.

[14] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[15] H. Tang, D. Xu, N. Sebe, Y. Wang, J. J. Corso, and Y. Yan. Multi-channel attention selection gan with cascaded semantic guidance for cross-view image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[16] F. Tonini, N. Dall'Asen, C. Beyan, and E. Ricci. Object-aware gaze target detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 21860–21869, 2023.

[17] H.-Y. Tseng, Q. Li, C. Kim, S. Alsisan, J.-B. Huang, and J. Kopf. Consistent view synthesis with pose-guided diffusion models. *arXiv preprint arXiv:2303.17598*, 2023.

[18] D. Watson, W. Chan, R. Martin-Brualla, J. Ho, A. Tagliasacchi, and M. Norouzi. Novel view synthesis with diffusion models. *arXiv preprint arXiv:2210.04628*, 2022.

[19] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7467–7477, 2020.

[20] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, et al. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024.

[21] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4578–4587, June 2021.

[22] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2223–2232, 2017.

APPENDIX A

Appendices

A.1 Code Repository and Resources

The code developed for this project is built on top of the existing repository of 3D Gaussian Splatting. Below is the link to the GitLab repository where the implementation and experiments discussed in this thesis can be found:

GitLab Repository Link: https://git.cs.bham.ac.uk/projects-2023-24/yxb340

To replicate the results and experiments from this thesis, please follow these steps :

1. Clone the repository
2. Setup Instructions: Ensure you have the necessary dependencies installed. Detailed instructions for installing dependencies can be found in the README file of the repository. Install the required packages using the provided requirements.txt or use Conda/virtual environments to avoid conflicts.
3. Running the Code: The repository contains scripts for both training and rendering. After setting up the environment, run the commands from README file to start training and rendering.

A.2 Dataset

The dataset used in the experiments is a custom video dataset captured using an iPhone. The dataset includes both exo-centric and ego-centric video pairs and is used to test the exo-to-ego view translation. The frames extracted from the videos are used as input to COLMAP to generate sparse point clouds.

The videos can be found in the link : Exo-Centric Input Videos

The dataset can be created by extracting frames from a video using the custom Python function we created, which is explained in detail in the README file.

Once we get those frames, they are passed as input to COLMAP to get sparse initialization for Gaussian Splatting. A sample of COLMAP output of the input frames can be found in the link: Sample COLMAP Output

A.3 Gaussian Splatting Algorithms

In this section, the algorithms used by 3D Gaussian Splatting used in the experiments are shown.

Algorithm 1 Optimization and Densification
w, h: width and height of the training images

$M \leftarrow$ SfM Points ▷ Positions
$S, C, A \leftarrow$ InitAttributes() ▷ Covariances, Colors, Opacities
$i \leftarrow 0$ ▷ Iteration Count
while not converged **do**
  $V, \hat{I} \leftarrow$ SampleTrainingView() ▷ Camera V and Image
  $I \leftarrow$ Rasterize(M, S, C, A, V) ▷ Alg. 2
  $L \leftarrow Loss(I, \hat{I})$ ▷ Loss
  $M, S, C, A \leftarrow$ Adam(∇L) ▷ Backprop & Step
  if IsRefinementIteration(i) **then**
    for all Gaussians (μ, Σ, c, α) **in** (M, S, C, A) **do**
      if $\alpha < \epsilon$ or IsTooLarge(μ, Σ) **then** ▷ Pruning
        RemoveGaussian()
      end if
      if $\nabla_p L > \tau_p$ **then** ▷ Densification
        if $\|S\| > \tau_S$ **then** ▷ Over-reconstruction
          SplitGaussian(μ, Σ, c, α)
        else ▷ Under-reconstruction
          CloneGaussian(μ, Σ, c, α)
        end if
      end if
    end for
  end if
  $i \leftarrow i + 1$
end while

Figure A.1: Optimization and Densification Algorithm

Algorithm 2 GPU software rasterization of 3D Gaussians
w, h: width and height of the image to rasterize
M, S: Gaussian means and covariances in world space
C, A: Gaussian colors and opacities
V: view configuration of current camera

function RASTERIZE(w, h, M, S, C, A, V)
  CullGaussian(p, V) ▷ Frustum Culling
  $M', S' \leftarrow$ ScreenspaceGaussians(M, S, V) ▷ Transform
  $T \leftarrow$ CreateTiles(w, h)
  $L, K \leftarrow$ DuplicateWithKeys(M', T) ▷ Indices and Keys
  SortByKeys(K, L) ▷ Globally Sort
  $R \leftarrow$ IdentifyTileRanges(T, K)
  $I \leftarrow \mathbf{0}$ ▷ Init Canvas
  for all Tiles t **in** I **do**
    for all Pixels i **in** t **do**
      $r \leftarrow$ GetTileRange(R, t)
      $I[i] \leftarrow$ BlendInOrder(i, L, r, K, M', S', C, A)
    end for
  end for
  return I
end function

Figure A.2: Fast Rasterization Algorithm

A.4 Gaze Detection Methodology

The Object-aware Gaze Target Detection method used in the gaze-based ego-centric rendering process is described in detail in the paper by Tonini et al. (2023). This method helps detect the gaze direction. A work-in-progress implementation of this method is available in the repository, along with results showing gaze direction and head position for exocentric frames, which are shown in Table. A.1

Table A.1: Gaze Direction and Head Position of Exo-Centric Images

Image Name	**Gaze Direction**	**Head Position**
frame_00001.png	[40.0, 38.0]	[697.2, 314.5, 743.18, 368.95]
frame_00002.png	[40.0, 38.0]	[705.3, 314.6, 750.07, 367.84]
frame_00003.png	[42.0, 37.0]	[715.0, 312.3, 763.4, 369.17]
frame_00004.png	[43.0, 37.0]	[725.3, 315.9, 770.07, 365.51]
frame_00005.png	[43.0, 37.0]	[728.1, 310.5, 775.29, 364.95]
frame_00006.png	[45.0, 37.0]	[725.4, 311.8, 768.96, 362.62]
frame_00007.png	[45.0, 36.0]	[715.0, 309.5, 763.4, 363.95]
frame_00008.png	[42.0, 36.0]	[708.0, 309.3, 756.4, 366.17]
frame_00009.png	[43.0, 36.0]	[707.9, 312.1, 757.51, 371.39]
frame_00010.png	[43.0, 36.0]	[710.9, 315.0, 760.51, 375.5]

A.5 Experimental Setup

All experiments were conducted on a system with the following configuration:

- GPU: NVIDIA RTX 3090
- RAM: 24 GB
- Software: Python 3.8, PyTorch 1.10
- Additional Packages: CUDA 11.8, COLMAP, OpenCV

Details of the specific software versions and environment variables can be found in the environment.yml file in the repository.

A.6 Output Files

The outputs generated from the experiments in this project include ego-centric rendered videos, images, and 3D Gaussian splats, which can be accessed through the following links:

- Ego-Centric Output Videos and Rendered Images
- 3D Gaussian Splats in .ply format

The 3D Gaussian splats are provided in the widely-used '.ply' format and can be visualized using tools like the **SIBR Viewer** or online platforms such as **Supersplat**, which can be found on the link: SuperSplat Online Viewer

To explore the outputs, download the provided '.ply' files and use the above-mentioned viewers for a detailed visual inspection of the 3D Gaussian splats.