MetaSCUT: Large-Scale Scene Simulation based on 3D Gaussian Splatting and Universal Physics Engine

Zhenyu Sun * Sitong Wang * Ye Pang * Zihao Chen * Chengkai Wang * Cen Chen *

{202264690427, 202264690083, 202264690373, 202264690182, 202264690434, chencen}@scut.edu.cn

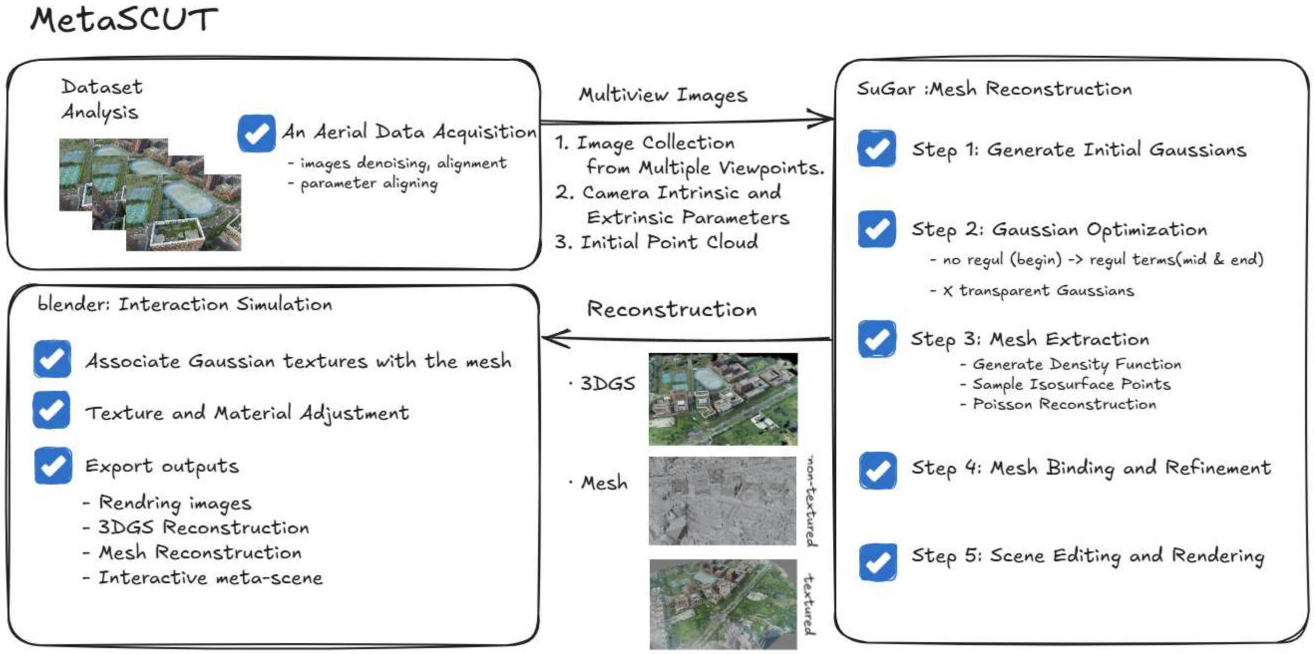

MetaSCUT
Dataset Analysis
An Aerial Data Acquisition
- images denoising, alignment
- parameter aligning
Multiview Images
1. Image Collection from Multiple Viewpoints.
2. Camera Intrinsic and Extrinsic Parameters
3. Initial Point Cloud
SuGar :Mesh Reconstruction
Step 1: Generate Initial Gaussians
Step 2: Gaussian Optimization
- no regul (begin) -> regul terms(mid & end)
- X transparent Gaussians
Step 3: Mesh Extraction
- Generate Density Function
- Sample Isosurface Points
- Poisson Reconstruction
Step 4: Mesh Binding and Refinement
Step 5: Scene Editing and Rendering
Reconstruction
· 3DGS
· Mesh
non-textured
textured
blender: Interaction Simulation
Associate Gaussian textures with the mesh
Texture and Material Adjustment
Export outputs
- Rendring images
- 3DGS Reconstruction
- Mesh Reconstruction
- Interactive meta-scene

Figure 1. **Overview of our MetaSCUT.** The framework begins with aerial data acquisition, involving image denoising, alignment, and parameter adjustment, which yield a preliminary dataset comprising multiple viewpoints, intrinsic and extrinsic camera parameters, and an initial point cloud. The SuGar mesh reconstruction module follows, generating initial Gaussians, optimizing their distribution, and extracting triangular meshes via density function generation, isosurface sampling, and Poisson reconstruction. Finally, Blender is used for scene rendering, associating Gaussian-based textures with the mesh, adjusting materials, and exporting rendering output as well as interaction demonstration.

Abstract

In the fields of digital twins and virtual reality, the accurate reconstruction and interaction of large-scale scenes have become key research focuses. In this paper, we present MetaSCUT, a framework covers digital twin reconstruction of campus (SCUT) and interactions within the reconstructed scene. For the campus reconstruction, we utilized a self-collected aerial dataset SCUTer, with each building in the campus divided into several zones, each containing hundreds of multi-view images. We applied a method based on gaussian splatting for mesh extraction, where we conducted a regularization term to align the Gaussian model with the scene surface and used Poisson reconstruction to efficiently extract an accurate mesh. For the interactive component, we primarily used Blender to implement interactive features, such as a physics-simulated vehicle in the virtual campus. Additionally, we explored the use of the Genesis physics simulation engine to control the movement of multiple robotic arms. In conclusion, we have developed an explorative cam-

pus scene that integrates large-scale reconstruction with interactive features, offering a novel solution for the precise reconstruction and interaction within large-scale environments.

1. Introduction

High-precision large-scale environment reconstruction [2, 5, 6] not only provides users with realistic visual experiences but also has significant application value in fields such as smart cities and digital campuses. However, achieving efficient and accurate reconstruction and interaction remains a major challenge due to the massive amount of data and the complex structures of scenes. Thus, exploring efficient methods for the reconstruction and interaction of large-scale environments is of great importance.

In this paper, we propose MetaSCUT, a superior framework that realizes large-scale scene reconstruction and interaction simulation.

For campus reconstruction, we used a self-collected aerial campus dataset SCUTer [19], which includes hundreds of multiview images of various campus building zones. To efficiently process this extensive dataset, we performed mesh reconstruction. Specifically, we refer to surface aligned Gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering (SuGaR) [2], which offers an innovative approach to efficient and precise mesh reconstruction from 3D Gaussian splatting (3DGS)[4]. SuGaR introduces a surface-aligned regularization term that ensures 3D Gaussians align well with the scene geometry, significantly improving the geometric accuracy and usability of extracted meshes. By leveraging this alignment, SuGaR facilitates efficient sampling on the real surface of the scene and employs Poisson reconstruction to extract high-quality meshes. Compared to traditional methods like Marching Cubes, this approach is faster, scalable, and capable of preserving fine details.

To handle the challenges of large-scale 3D scenes, SuGaR provides an optional refinement strategy that binds Gaussians to the surface of the extracted mesh and jointly optimizes both through Gaussian splatting rendering. This joint optimization ensures seamless integration of the Gaussians and the mesh, enabling downstream editing, sculpting, rigging, and rendering using conventional tools like Blender and Unreal Engine. Moreover, the method achieves editable mesh reconstruction within minutes, providing a substantial performance advantage over state-of-the-art approaches based on Neural SDFs [18], which typically require hours of computation.

Building on the digital twin reconstruction of large-scale environments, further enhancing the interactivity of virtual scenes is crucial for improving user experience and expanding application domains. To enable rich interactive functionalities within the reconstructed campus scene, we aim to create a vibrant and explorable virtual world through dynamic animations and rich interactive logic. For this purpose, we chose Blender as the core development tool, which perfectly meets our needs with its powerful animation creation, interactive features, and efficient workflows.Blender is a comprehensive open-source 3D creation tool that provides an efficient development experience with its robust animation and interactive design features. In the virtual campus scene, Blender's capabilities in rigging, keyframe animation, and non-linear animation editing enable us to realize robotic arm movements, character actions, and dynamic scene effects. Furthermore, Python scripting allows us to design user interaction logic, making the virtual environment more vivid and intelligent.In the virtual campus scene, we used Blender's animation editor to design various dynamic effects, including robotic arm operations, object motion paths within the scene, and changes in environmental lighting and shadows. Additionally, with Python scripting, we implemented interactive features for users to trigger animations, adjust perspectives, and control scene objects.

To further enrich the interactivity of the virtual campus, we also adopted the ProbTalk3D method [16]. This approach employs a VQ-VAE-based probabilistic 3D animation synthesis technique to generate diverse interactive content based on input information. By incorporating this technology, we introduced interactive robotic arms into the campus scene, allowing users to manipulate objects through robotic arm controls, thereby enhancing the scene's interactivity and immersion. Additionally, we referenced the ARCHITECT framework [14] to create vivid and interactive 3D scenes. ARCHITECT utilizes diffusion-based 2D image inpainting techniques to construct complex and realistic 3D environments. Through a hierarchical and iterative inpainting process, it transforms 2D images into 3D point clouds, adding more detail and interactive elements to our campus scene.

2. Digital Twin

2.1. Urban Scene Reconstruction

Urban scene reconstruction has evolved significantly over the years, driven by advancements in 3D representation and rendering techniques. Early approaches focused on sparse point cloud representations, while recent methods leverage neural volumetric rendering and Gaussian-based representations to achieve higher fidelity and efficiency. This section reviews the progression of these techniques, from point-based methods to NeRF-based approaches and the latest Gaussian-based innovations. Point-based methods, i.e. [9, 10, 15, 17], represent scenes using sparse point clouds, enabling rapid 3D scene generation. Point-E [9] is particularly suited for coarse modeling due to its lightweight representation and low storage requirements. However, its reliance on sparse point clouds limits its ability to capture fine details and complex geometries, making it less suitable for high-precision urban reconstruction tasks.

Neural Radiance Fields (NeRF) [7] revolutionized neural rendering by modeling complex lighting and geometry through volumetric rendering with neural networks. Although highly effective for small-scale scenes, its computational cost and scalability limitations restrict its use in large-scale urban environments, prompting the development of various extensions. To address the challenges, significant efforts have been devoted to related research [8, 12, 13]. Mega-NeRF [13] introduces a spatial partitioning strategy, enabling parallel training of different segmented scenes and improving scalability for large-scale scenes. However, the fusion of partitions and reliance on neural volumetric representations still result in computational bottlenecks. Mip-NeRF 360 [12] further extends NeRF by incorporating multi-scale volumetric sampling, which mitigates issues such as light leakage and improves performance in large, complex outdoor scenes. Despite these advancements, it remains computationally expensive and lacks real-time rendering capabilities, limiting its practicality for dynamic urban applications.

Recent advancements in Gaussian-based representations, such as 3D Gaussian Splatting (3DGS) [4], have introduced a new paradigm for urban scene reconstruction. CityGaussian [5, 6] leverages 3DGS to achieve efficient representation and rendering of large-scale scenes. By employing a lightweight 3D Gaussian-based representation, combined with block-based training, adaptive data selection, and multi-level detail (LoD) management, it enables highly efficient real-time rendering and scalable performance.

2.2. Mesh Extraction from 3D Gaussian Splatting

2.3. Simulation Platform

In the context of using Genesis, an advanced physical simulation platform, it is essential to compare it with related platforms to fully understand its unique position in robotics and AI research. While Genesis offers a unified physics engine, cross-domain versatility, and efficient automated data generation, other platforms also excel in specific domains. These comparisons can provide deeper insights into the core features of Genesis while clarifying the scenarios and complementary roles of other tools.

2.3.1. Isaac Sim

Isaac Sim is a powerful simulation platform developed by NVIDIA, designed for robotics and AI research. It leverages NVIDIA's Omniverse platform to deliver high-fidelity, physically accurate, and scalable simulations. The tool supports developers in designing, simulating, testing, and training robots and autonomous systems in virtual environments that replicate real-world conditions. Below are the comparison between Issac Sim and Genesis:

- **Hardware Dependency.** Genesis is cross-platform and hardware-agnostic, while Isaac Sim is GPU-intensive and optimized for NVIDIA RTX GPUs.
- **Rendering Fidelity.** Isaac Sim excels with exceptional ray tracing support for photorealistic rendering, while Genesis offers high-quality but less photorealistic rendering.
- **Synthetic Data Generation.** Both platforms provide automated synthetic data generation; Isaac Sim focuses on high-quality, vision-driven data.
- **Target Use Case.** Genesis supports general-purpose robotics and AI research, while Isaac Sim is tailored for vision-based AI and robotics visualization.

2.3.2. Gazebo

Gazebo is an open-source robotics simulator widely used in research and industry for testing and developing robotic systems in complex, realistic environments. It excels at simulating populations of robots in both indoor and outdoor scenarios with high accuracy and efficiency. The differences between Genesis and Gazebo are illustrated as below:

- **Physics Modeling:** Genesis supports complex physical interactions, such as soft bodies and fluids, while Gazebo focuses on rigid body dynamics.
- **Ease of Use.** Genesis emphasizes simplified environment creation, whereas Gazebo may require more configuration and setup.
- **Cross-Platform Support.** Genesis is fully cross-platform, while Gazebo's optimization is more limited.
- **Target Use Case.** Gazebo is ideal for multi-robot simulations and dynamic environments, while Genesis has a broader scope in robotics and AI research.

2.3.3. RFUniverse

RFUniverse is a multiphysics simulation platform designed for embodied AI research. It uses Unity and a gRPC-based client-server framework to facilitate interaction with virtual environments. Specific comparisons with Genesis are as follows:

- **Core Engine.** Genesis employs a custom physics engine, whereas RFUniverse relies on Unity.
- **Multi-Physics Support.** Genesis seamlessly integrates multi-physics, while RFUniverse depends on Unity plug-ins.
- **Interactivity.** RFUniverse offers highly interactive environments via Unity assets, while Genesis provides robust but less Unity-like interactivity.
- **Target Use Case.** RFUniverse is focused on embedded AI and manipulation tasks, while Genesis supports a wider range of robotics and AI research.

2.3.4. Blender

Blender is an open-source 3D computer graphics software whose powerful features make it an essential tool in projects like virtual reality and digital twins. The core principles

Work Feature	SuGaR	CityGaussian	Mega-NeRF	Mip-NeRF 360	Point-E
Representation	3DGS (Surface Alignment)	3DGS	NeRF	NeVF	Point Cloud
Applicable Scale	Large-Scale, Urban	Large-Scale, Urban	Large-Scale	Large-Scale, Outdoor	Coarse
Training Efficiency	High	High	Medium	Low	High
Real-time Rendering	Strong	Strong	Moderate	Weak	Strong
Detail Management (LoD)	Multi-Level Detail Management	Multi-Level Detail Management	Supported	Supported	None
Storage Requirement	Low	Low	High	High	Low
Mesh Reconstruction	✓	-	-	-	-

Table 1. **Comparative Analysis of Urban Scene Reconstruction Methods.** Compared to CityGaussian, SuGaR enhances detail management through the integration of Poisson Reconstruction with Gaussian Splatting, offering multi-level detail that is better aligned with surface geometries. Additionally, SuGaR is the only method in the comparison that supports mesh reconstruction. While methods like Mega-NeRF and Mip-NeRF 360 support large-scale reconstructions, they suffer from higher storage requirements and lower training efficiency. Point-E, though efficient with strong real-time rendering, provides only coarse representations without multi-level detail management.

of Blender are based on 3D computer graphics, supporting various modeling techniques such as mesh modeling, curve modeling, and sculpting. Its rendering engines (such as Cycles and Eevee) can generate high-quality images, supporting advanced rendering technologies like ray tracing and global illumination. Blender's simulation capabilities cover various aspects such as physics, fluid dynamics, smoke, and cloth, providing reliable support for the creation and simulation of complex scenes. Additionally, Blender supports Python scripting, allowing users to extend functionality and automate tasks, greatly improving work efficiency.

2.3.5. Genesis

Genesis is a state-of-the-art open-source physics simulation engine that employs efficient differentiable simulation methods to model the dynamic behavior of articulated bodies [11]. This approach combines the dynamics of articulated bodies with deep learning frameworks to enable gradient-based optimization, significantly improving the efficiency and accuracy of simulations. Moreover, Genesis extends the scale of robotic learning through generative models by automating the generation of 3D assets, task descriptions, task decomposition, and reward functions, thereby reducing human involvement [3]. This automated strategy generation and learning mechanism provides a feasible path for scaling robotic skill learning in simulated environments.

3. Experiment Method

To achieve efficient 3D reconstruction and dynamic interaction, this method is divided into four primary components: data acquisition and preprocessing, SuGaR modeling, Blender integration, and testing and validation. Each component is described in detail below.

3.1. Data Acquisition and Preprocessing

To ensure high-quality input for 3D reconstruction, aerial data is collected and preprocessed. These steps establish the foundation for reliable model generation using SuGaR.

Aerial Dataset. The custom dataset is captured by DJI Zenmuse L2 [1]. The dataset contains 368 aerial images, with high resolution 5472×3648 for each images, meticulously documenting the scenes of the Guangzhou International Campus of South China University of Technology.

3.2. 3DGS-based Mesh Reconstruction.

SuGaR provides an efficient pipeline for 3D reconstruction by aligning 3D Gaussian distributions to the scene surfaces, thus offering a better .

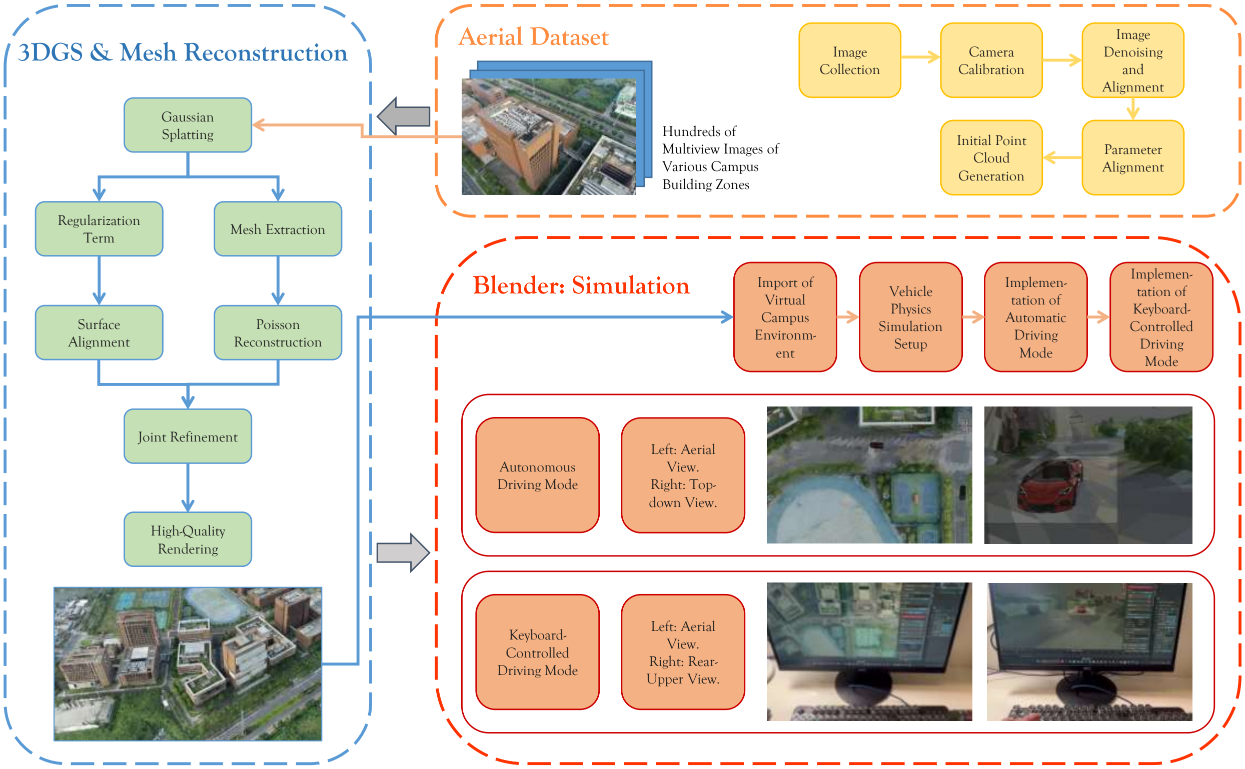

3DGS & Mesh Reconstruction
Gaussian Splatting
Regularization Term
Mesh Extraction
Surface Alignment
Poisson Reconstruction
Joint Refinement
High-Quality Rendering
Aerial Dataset
Hundreds of Multiview Images of Various Campus Building Zones
Image Collection
Camera Calibration
Image Denoising and Alignment
Parameter Alignment
Initial Point Cloud Generation
Blender: Simulation
Import of Virtual Campus Environm-ent
Vehicle Physics Simulation Setup
Implemen-tation of Automatic Driving Mode
Implemen-tation of Keyboard-Controlled Driving Mode
Autonomous Driving Mode
Left: Aerial View. Right: Top-down View.
Keyboard-Controlled Driving Mode
Left: Aerial View. Right: Rear-Upper View.

Figure 2. **Pipeline of our MetaSCUT, which integrates dataset analysis, SuGar-based mesh reconstruction, and Blender-assisted interaction simulation.** The process begins with aerial data acquisition, including denoising, alignment, and parameter adjustment, to generate an initial point cloud and camera parameters. SuGar mesh reconstruction involves five key steps: generating initial Gaussian representations, optimizing the Gaussians with regularization terms, extracting the mesh using density functions and Poisson reconstruction, refining and binding the mesh, and rendering the scene. Finally, the output mesh is enriched in Blender with Gaussian-based textures, material adjustments, and is exported as static images or videos.

3.2.1. Input processing and Configurations

SuGaR accepts multiview images as input, which should cover various angles of the scene to ensure reconstruction completeness and accuracy. High-resolution images are recommended, with consistent lighting conditions to avoid color discrepancies in the model. Input images are typically stored in standard formats such as JPEG or PNG and undergo preprocessing (e.g., undistortion, normalization) to meet SuGaR's quality requirements.

SuGaR offers a series of Python scripts to execute the complete 3D reconstruction process. Users can run the specific script to perform the entire pipeline, including Gaussian initialization, optimization, and mesh extraction. For more specific tasks, example python scripts are given as well.Key parameters, such as image paths, optimization iterations, and output formats, are adjustable via configuration files

3.2.2. SuGaR Reconstruction Workflow

The SuGaR pipeline introduces a structured approach for extracting highly detailed and editable 3D meshes from Gaussian Splatting representations. Below, the key steps and theoretical foundations are outlined:

Camera Estimation The pipeline begins with *Structure-from-Motion (SfM)* to estimate the camera extrinsic parameters and generate a sparse point cloud. COLMAP, a state-of-the-art SfM tool, is used for feature matching across multiview images, providing accurate camera poses and sparse geometric data. This initialization step forms the basis for subsequent Gaussian optimization.

Gaussian Initialization and Optimization The set of Gaussians is initialized from the sparse point cloud generated by COLMAP. Each Gaussian g is represented by:

- Mean position μ_g,
- Covariance Σ_g, parameterized by scaling vector s_g and rotation quaternion q_g,
- Opacity $\alpha_g \in [0, 1]$, and
- Spherical harmonics coefficients for color emission.

The Gaussians are optimized using a loss function that aligns rendered views of the Gaussians with the input images. New Gaussians are added iteratively to better capture the scene geometry and texture.

Surface-Aligned Regularization. To align the Gaussians with the scene surface, SuGaR introduces a novel regularization term:

$$R = \frac{1}{|P|} \sum_{p \in P} |f(p) - f^{\text{ideal}}(p)|, \tag{1}$$

where $f(p)$ is the signed distance function (SDF) derived from the current Gaussian distribution, and $f^{\text{ideal}}(p)$ is the ideal SDF for a perfectly aligned surface. This encourages Gaussians to distribute uniformly across the surface and minimizes overlap.

The Gaussian density function is defined as:

$$d(p) = \sum_{g} \alpha_g \exp\left(-\frac{1}{2}(p-\mu_g)^T \Sigma_g^{-1}(p-\mu_g)\right), \tag{2}$$

and the SDF is derived as:

$$f(p) = \pm s_g^* \sqrt{-2\log(d(p))}. \tag{3}$$

Mesh Extraction Using the aligned Gaussians, SuGaR employs a scalable sampling method to identify points on the density level set:

$$\{p \mid d(p) = \lambda\}, \tag{4}$$

where λ is a threshold parameter. The sampled points and their normals are passed to Poisson Surface Reconstruction, producing a high-fidelity triangle mesh. This approach avoids the pitfalls of traditional methods like Marching Cubes, which struggle with sparsely distributed density functions.

Mesh and Gaussian Refinement (Optional) An optional refinement step jointly optimizes the mesh and the Gaussians by binding Gaussians to the mesh triangles. Each triangle is assigned a set of thin 3D Gaussians, whose parameters are optimized to enhance rendering fidelity. The refinement aligns Gaussians with the mesh surface while preserving editability and maintaining high-quality rendering.

Textured Mesh Generation The final mesh can be textured using the original input images, enabling compatibility with traditional 3D software like Blender, Unity, and Unreal Engine. The Gaussian-bound meshes offer superior rendering quality compared to standard UV-textured meshes.

3.2.3. Mesh Simplification and textures Mapping

SuGaR-generated meshes can be simplified using Blender optimized with libraries such as MeshLab or CGAL. Additionally, to enhance visual quality, textures from the original images can be mapped onto the generated mesh. SuGaR supports textured mesh extraction and embeds texture data into standard format files (e.g., OBJ, FBX), enabling seamless import and rendering in Blender.

3.3. Blender Import and Dynamic Interaction

To maximize the usability of SuGaR-generated models, this section explores their integration into Blender and the implementation of dynamic interaction functionalities.

3.3.1. Model Import

This subsection discusses the process of importing SuGaR-generated 3D models into Blender, including format compatibility and preparation for further use.

3.3.2. Design of Dynamic Interactions

Dynamic interactions in Blender are implemented through script development and user interface design. This subsection outlines the approach used to define interaction goals and achieve seamless functionality.

3.3.3. Optimization and Performance Tuning

Given the complexity of models and the demand for real-time performance, this subsection describes strategies for optimizing model rendering and interaction logic.

4. Experimental Results and Analysis

4.1. Scene Reconstruction

In our experiments, we used SuGaR [2] to complete the digital twin reconstruction of the campus, South China University of Technology, Guangzhou International Campus.

The results obtained using the CityGaussian method are shown in Figure **??**, while those generated using the SuGaR method are presented in Figure **??**. As illustrated, the models produced by CityGaussian exhibit significant shortcomings in clarity and detail, with blurred and distorted edges and structures. In contrast, the SuGaR method demonstrates superior performance in capturing edge details, restoring textures, and accurately representing overall structures. The models generated by SuGaR are characterized by high clarity, rich details, and a faithful reproduction of the campus environment.

The primary reason for this performance difference lies in SuGaR's introduction of a regularization term, which allows Gaussian distributions to align more accurately with the surfaces of real-world scenes. This effectively reduces reconstruction errors caused by factors such as noise, occlusion, and uneven sampling. Additionally, SuGaR leverages Poisson reconstruction techniques to efficiently extract precise meshes. This capability is particularly critical for our task, as the campus scene comprises a large number of elements, including buildings, roads, and vegetation, requiring highly efficient and accurate mesh generation. Furthermore, SuGaR's Gaussian scattering model enables a more precise representation of geometric and texture information in the scene. By aligning the Gaussian distributions with the surfaces, the method captures the true shapes and details

Method	3D Representation	VDQ	Rendering Quality			Artifacts	Holes
			PSNR ↑	SSIM ↑	LPIPS ↓		
CityGS	3DGS	-	13.153	0.247	0.635	✓	✓
SuGaR	3DGS & Mesh Hybrid	0.1	22.592	0.668	0.346	None	✓
SuGaR(ours)	**3DGS & Mesh Hybrid**	**0**	**22.667**	**0.668**	**0.347**	**None**	**None**

Table 2. **Quantitative comparison of the reconstruction performance between different methods [2, 6].** We evaluate the CityGS, SuGaR, and the fine-tuned SuGaR on Rendering Quality aspects, including PSNR, SSIM and LPIPS. With the proper hyper-parameter settings, Sugar module outperforms the vanilla version and CityGS on rendering quality, while producing 3D reconstruction results with less artifacts and empty holes, laying a solid foundation for subsequent simulated interactions.

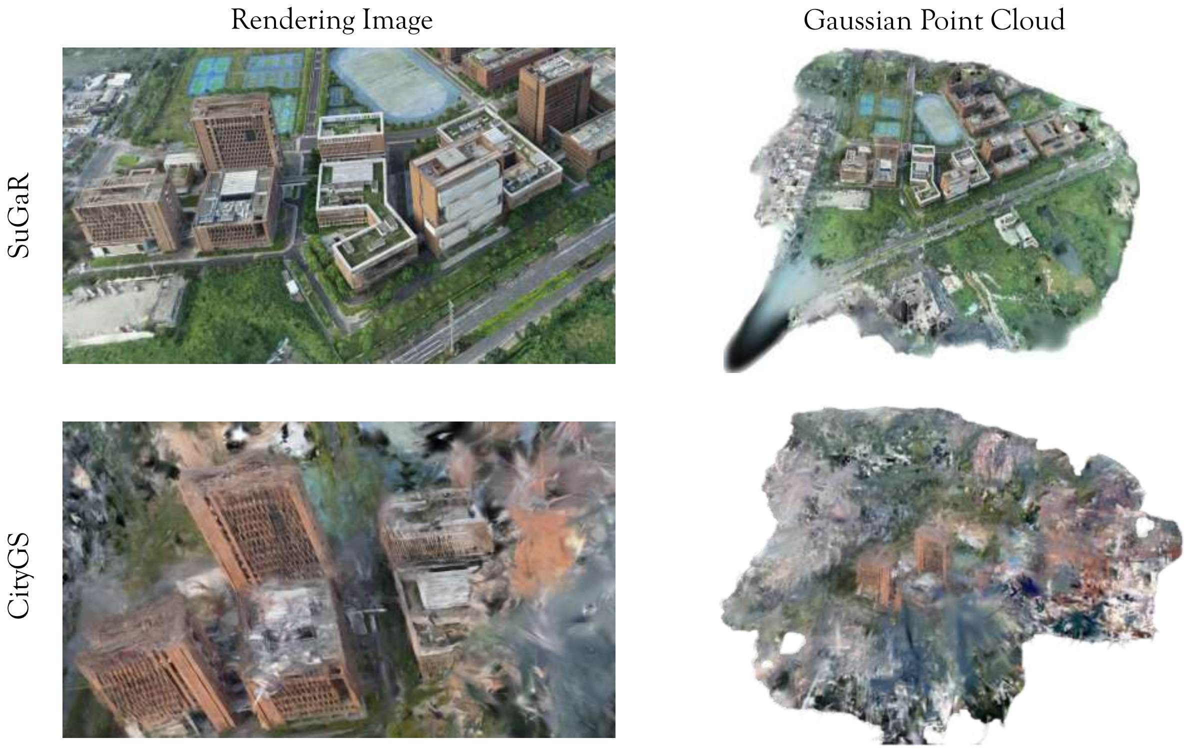


Figure 3. **Qualitative comparison of reconstruction results obtained Using the CityGS and SuGaR.** As demonstrated in Figure 3 , SuGaR performs more powerfully than CityGS, both in rendering images as well as the 3DGS representaion aspects.

of objects more effectively. On the other hand, while CityGaussian is also based on Gaussian primitives, its focus on regional training may overlook the completeness of global structures and the accuracy of finer details.

Through comparative analysis, we have identified SuGaR as the primary technical solution for campus reconstruction. It provides a high-quality 3D scene foundation, which is essential for the subsequent implementation of interactive functionalities within the virtual campus.

4.2. Interaction Simulation

We utilized Blender software to perform the physical simulation of vehicles within the previously constructed virtual campus environment. Using Blender, we were able to import vehicle models into the virtual campus scene and precisely configure their physical properties, such as mass, friction coefficient, and collision boundaries, to simulate the dynamic behavior of real-world vehicles.

Two vehicle driving modes were implemented within the virtual campus: autonomous driving and keyboard-controlled

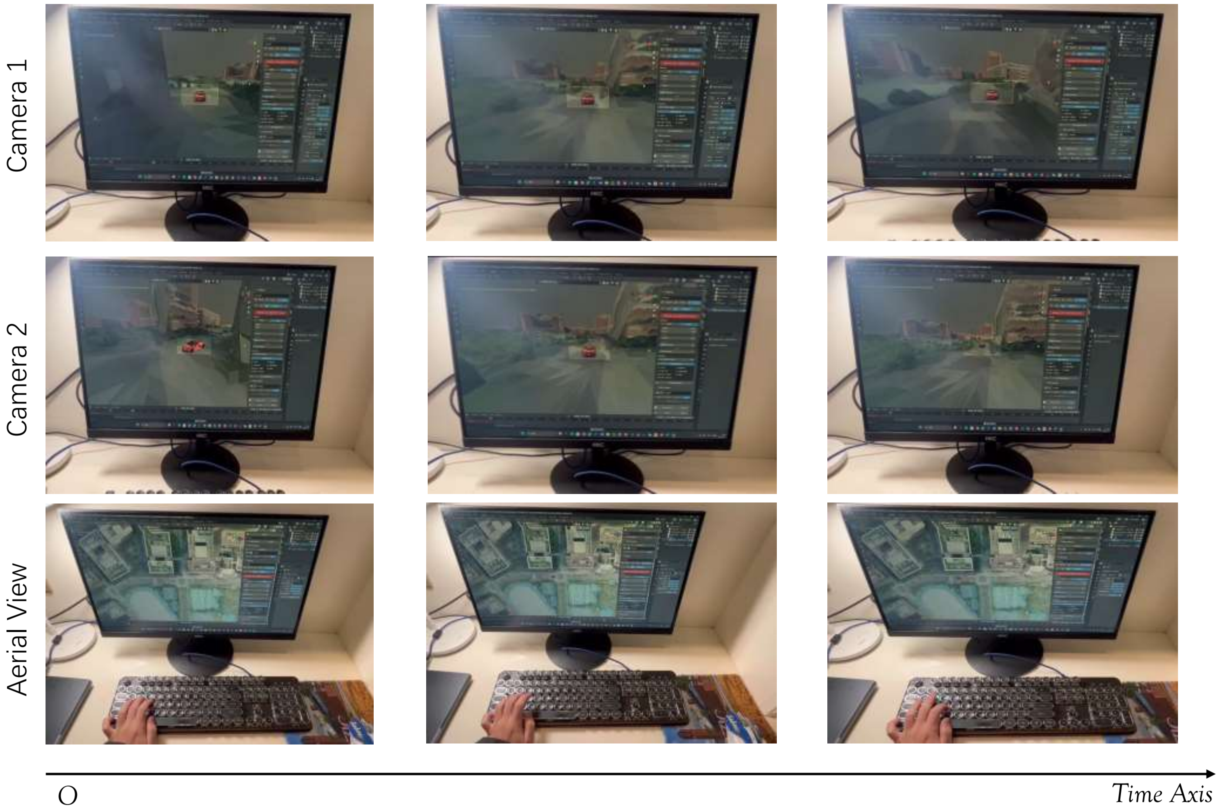


Figure 4. **Keyboard interaction with our MetaSCUT.** We can use the $W\ S\ A\ D$ to control the vehicle to move forward, move backward, turn left and turn right respectively.

driving. In the autonomous driving mode, vehicles navigate autonomously along a predefined path within the campus. We designed a path planning algorithm for the vehicles, defining navigation waypoints, and employed curve interpolation to generate smooth driving trajectories, allowing the vehicles to travel along the specified routes seamlessly. During the simulation, the vehicles also automatically adjust their steering angles based on the curvature of the path, ensuring smooth navigation. The simulation results of autonomous driving are shown in Figure 5, where an aerial view clearly illustrates the driving paths of two vehicles on the campus roads.

Additionally, to enhance the interactivity of the virtual campus, we implemented a keyboard-controlled driving mode. In this mode, users can control the direction, acceleration, and braking of the vehicles in real time via the keyboard. Users can drive the vehicles from either a first-person or third-person perspective, exploring various areas and buildings within the campus. The vehicle driving feature provides an immersive experience for users and adds dynamic elements to the virtual campus.

Videos showcasing both vehicle driving modes are available on the project's homepage for a clearer demonstration.

5. Discussion

We also utilized the Genesis physics simulation engine to simulate and validate the motion control of robotic arms. With its high-precision physics computation capabilities and flexible simulation environment, Genesis provides strong support for the dynamic simulation of complex mechanical systems such as robotic arms.

First, we implemented the motion control of a single robotic arm, as shown in Figure 6a. In the simulation scenario, the robotic arm successfully performed operations such as object grasping, transportation, and placement. The joint movements of the robotic arm were smooth, and the end-effector accurately reached the predetermined positions, demonstrating excellent trajectory tracking performance. Subsequently, we extended the simulation to include collaborative motion control of multiple robotic arms, as illustrated in Figure 6b. Multiple robotic arms operated synchronously within the same simulation environment, coordinating with each other to complete complex tasks.

However, since Genesis is still in its early development

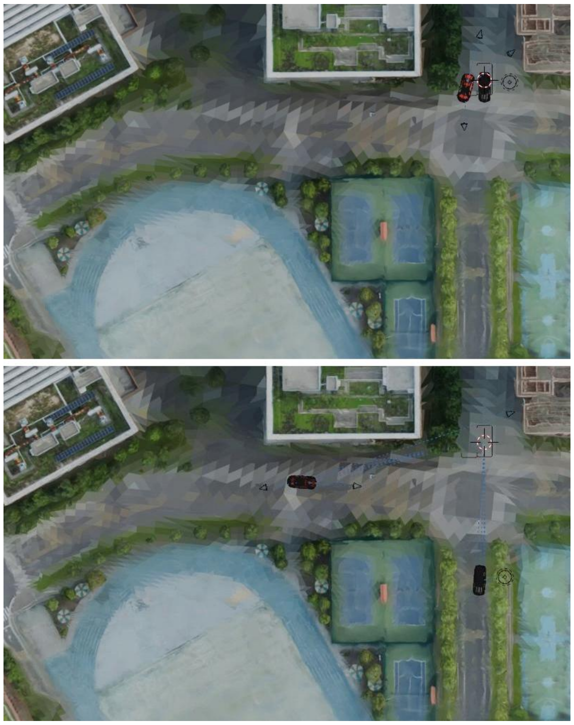
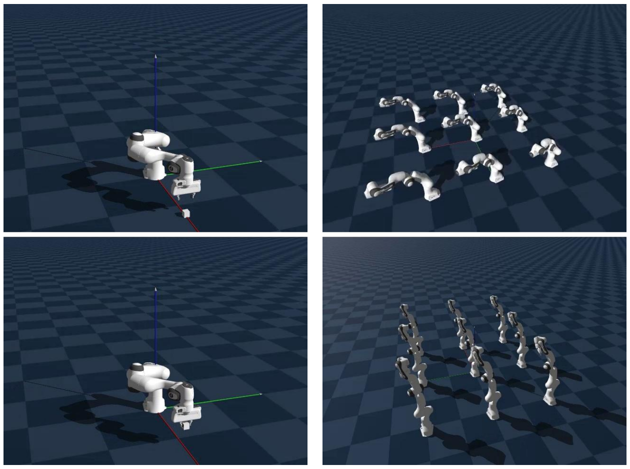


Figure 5. **Autonomous Vehicles within the Virtual Campus.** We can describe the



(a) Single Robotic Arm Grasping an Object.

(b) Collaborative Motion of Multiple Robotic Arms.

Figure 6. **Simulation of Robotic Arm Movements in Genesis.**

stages, it currently lacks support for loading and rendering external complex scenes. As a result, it cannot be directly applied to our virtual campus environment, which represents a limitation of this experiment. In the future, as Genesis continues to evolve and mature, it will enable realistic physical simulations and human-machine interactions involving robotic arms within the virtual campus. Potential applications include equipment operation in virtual laboratories, robotic teaching, and more. This advancement will significantly enhance the interactivity and immersion of the virtual campus, providing users with a richer experience.

6. Conclusion

Conclusion The MetaSCUT framework offers a robust solution for large-scale scene simulation, combining accurate digital twin reconstruction with interactive features. Utilizing a Gaussian splatting-based method, it achieves high-quality mesh reconstruction and efficient scene rendering. Blender integration enables dynamic interactions, enhancing user engagement. The SuGaR module's regularization and Poisson reconstruction techniques outperform traditional methods, providing detailed and accurate models. Looking ahead, the development of the Genesis physics simulation engine presents exciting opportunities to further enhance MetaSCUT. As Genesis matures, its advanced simulation capabilities will enable more realistic and interactive virtual environments, enriching the user experience. Future work will focus on integrating Genesis to expand the framework's capabilities and applications, ensuring MetaSCUT remains at the forefront of digital twin and virtual reality technologies.

References

[1] DJI. Dji enterprise, 2025. 4

[2] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775*, 2023. 2, 6, 7

[3] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models. *arXiv preprint arXiv:2310.18308*, 2023. 4

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

[5] Yang Liu, Chuanchen Luo, Zhongkai Mao, Junran Peng, and Zhaoxiang Zhang. Citygaussianv2: Efficient and geometrically accurate reconstruction for large-scale scenes. *arXiv preprint arXiv: 2411.00771*, 2024. 2, 3

[6] Yang Liu, Chuanchen Luo, Lue Fan, Naiyan Wang, Junran Peng, and Zhaoxiang Zhang. Citygaussian: Real-time high-quality large-scale scene rendering with gaussians. In *ECCV*, 2025. 2, 3, 7

[7] Ben Mildenhall, Pratul P. Srinivasan, Markatteri Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *CVPR*, 2020. 3

[8] Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, Peter Hedman, Ricardo Martin-Brualla, and Jonathan T. Barron. Multi-NeRF: A Code Release for Mip-NeRF 360, Ref-NeRF, and RawNeRF, 2022. 3

[9] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point·e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022. 2

[10] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[11] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Efficient differentiable simulation of articulated bodies. *arXiv preprint arXiv:2109.07719*, 2021. 4

[12] Markatteri Tancik, Pratul P. Srinivasan, Jonathan T. Barron, Ben Mildenhall, and Ren Ng. Mip-nerf 360: A multiscale neural radiance field for 360° view synthesis. In *CVPR*, 2023. 3

[13] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. *arXiv preprint arXiv:2112.10703*, 2022. 3

[14] Yian Wang, Xiaowen Qiu, Jiageng Liu, Zhehuan Chen, Jiting Cai, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. Architect: Generating vivid and interactive 3d scenes with hierarchical 2d inpainting. *arXiv preprint arXiv:2411.09823*, 2024. 2

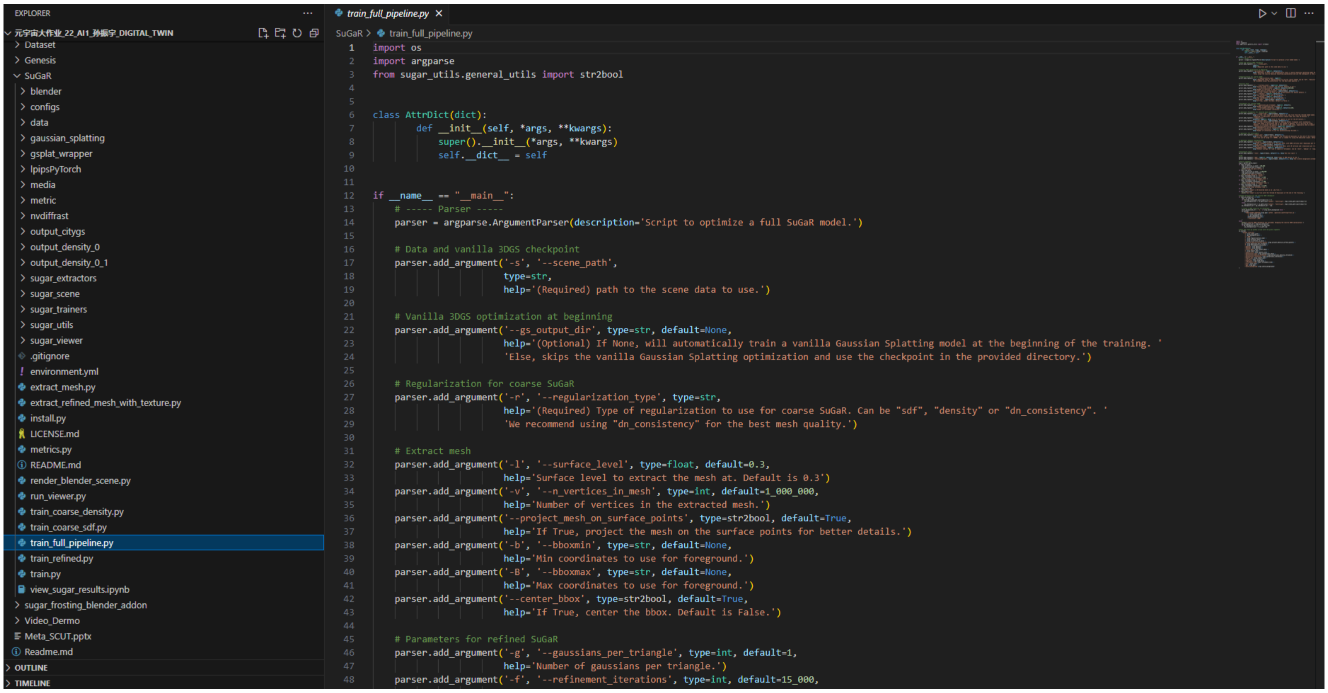
[15] Xi Wei, Zhiqiang Wei, et al. Voxel-based 3d point cloud processing for urban scene reconstruction. *IJCV*, 2019. 2

[16] Sichun Wu, Kazi Injamamul Haque, and Zerrin Yumak. Probtalk3d: Non-deterministic emotion controllable speech-driven 3d facial animation synthesis using vq-vae. *arXiv preprint arXiv:2409.07966*, 2024. 2

[17] Jianxiong Xiao, Yifan Zhang, Xialei Liu, and J. Sun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2

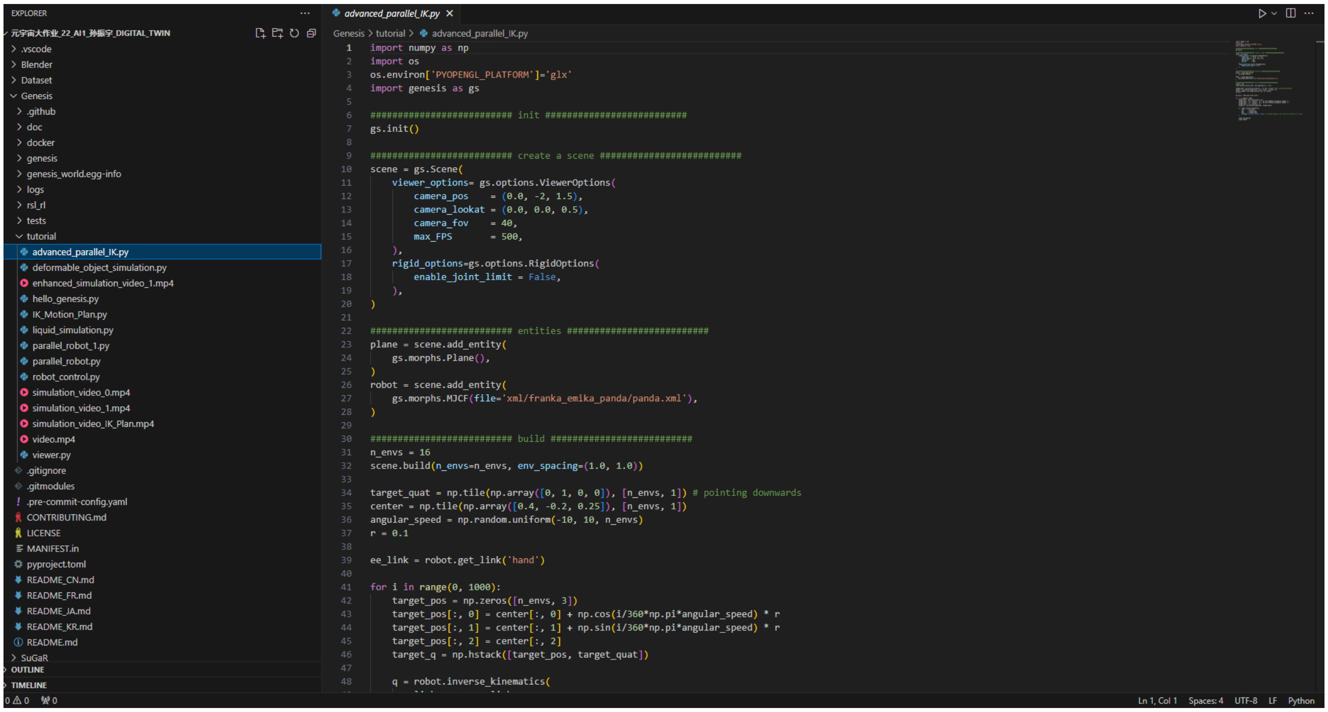
[18] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis, 2023. 2

[19] Xiaohan Zhang, Yukui Qiu, Zhenyu Sun, and Qi Liu. Aerial-nerf: Adaptive spatial partitioning and sampling for large-scale aerial rendering. *arxiv preprint arxiv:2405.06214*, 2024. 2



```python
import os
import argparse
from sugar_utils.general_utils import str2bool


class AttrDict(dict):
        def __init__(self, *args, **kwargs):
            super().__init__(*args, **kwargs)
            self.__dict__ = self


if __name__ == "__main__":
    # ----- Parser -----
    parser = argparse.ArgumentParser(description='Script to optimize a full SuGaR model.')

    # Data and vanilla 3DGS checkpoint
    parser.add_argument('-s', '--scene_path',
                        type=str,
                        help='(Required) path to the scene data to use.')

    # Vanilla 3DGS optimization at beginning
    parser.add_argument('--gs_output_dir', type=str, default=None,
                        help='(Optional) If None, will automatically train a vanilla Gaussian Splatting model at the beginning of the training. '
                        'Else, skips the vanilla Gaussian Splatting optimization and use the checkpoint in the provided directory.')

    # Regularization for coarse SuGaR
    parser.add_argument('-r', '--regularization_type', type=str,
                        help='(Required) Type of regularization to use for coarse SuGaR. Can be "sdf", "density" or "dn_consistency". '
                        'We recommend using "dn_consistency" for the best mesh quality.')

    # Extract mesh
    parser.add_argument('-l', '--surface_level', type=float, default=0.3,
                        help='Surface level to extract the mesh at. Default is 0.3')
    parser.add_argument('-v', '--n_vertices_in_mesh', type=int, default=1_000_000,
                        help='Number of vertices in the extracted mesh.')
    parser.add_argument('--project_mesh_on_surface_points', type=str2bool, default=True,
                        help='If True, project the mesh on the surface points for better details.')
    parser.add_argument('-b', '--bboxmin', type=str, default=None,
                        help='Min coordinates to use for foreground.')
    parser.add_argument('-B', '--bboxmax', type=str, default=None,
                        help='Max coordinates to use for foreground.')
    parser.add_argument('--center_bbox', type=str2bool, default=True,
                        help='If True, center the bbox. Default is False.')

    # Parameters for refined SuGaR
    parser.add_argument('-g', '--gaussians_per_triangle', type=int, default=1,
                        help='Number of gaussians per triangle.')
    parser.add_argument('-f', '--refinement_iterations', type=int, default=15_000,
```

Figure 7. Part of Code of SuGaR.



```python
import numpy as np
import os
os.environ['PYOPENGL_PLATFORM']='glx'
import genesis as gs

########################## init ##########################
gs.init()

########################## create a scene ##########################
scene = gs.Scene(
    viewer_options= gs.options.ViewerOptions(
        camera_pos    = (0.0, -2, 1.5),
        camera_lookat = (0.0, 0.0, 0.5),
        camera_fov    = 40,
        max_FPS       = 500,
    ),
    rigid_options=gs.options.RigidOptions(
        enable_joint_limit = False,
    ),
)

########################## entities ##########################
plane = scene.add_entity(
    gs.morphs.Plane(),
)
robot = scene.add_entity(
    gs.morphs.MJCF(file='xml/franka_emika_panda/panda.xml'),
)

########################## build ##########################
n_envs = 16
scene.build(n_envs=n_envs, env_spacing=(1.0, 1.0))

target_quat = np.tile(np.array([0, 1, 0, 0]), [n_envs, 1]) # pointing downwards
center = np.tile(np.array([0.4, -0.2, 0.25]), [n_envs, 1])
angular_speed = np.random.uniform(-10, 10, n_envs)
r = 0.1

ee_link = robot.get_link('hand')

for i in range(0, 1000):
    target_pos = np.zeros([n_envs, 3])
    target_pos[:, 0] = center[:, 0] + np.cos(i/360*np.pi*angular_speed) * r
    target_pos[:, 1] = center[:, 1] + np.sin(i/360*np.pi*angular_speed) * r
    target_pos[:, 2] = center[:, 2]
    target_q = np.hstack([target_pos, target_quat])

    q = robot.inverse_kinematics(
```

Figure 8. Part Code of Genesis.

A. Appendix

Part of the related code is attached here.

For the complete code and the video demonstration results, you can find them in the supplementary material we send to you.