

Proc-GS: Procedural Building Generation for City Assembly with 3D Gaussians

Yixuan Li¹, Xingjian Ran², Lining Xu¹, Tao Lu³, Mulin Yu², Zhenzhi Wang¹

Yuanbo Xiangli⁴, Dahua Lin^{1,2}, Bo Dai^{5,2}✉

¹ The Chinese University of Hong Kong ² Shanghai Artificial Intelligence Laboratory

³ Brown University ⁴ Cornell University ⁵ The University of Hong Kong

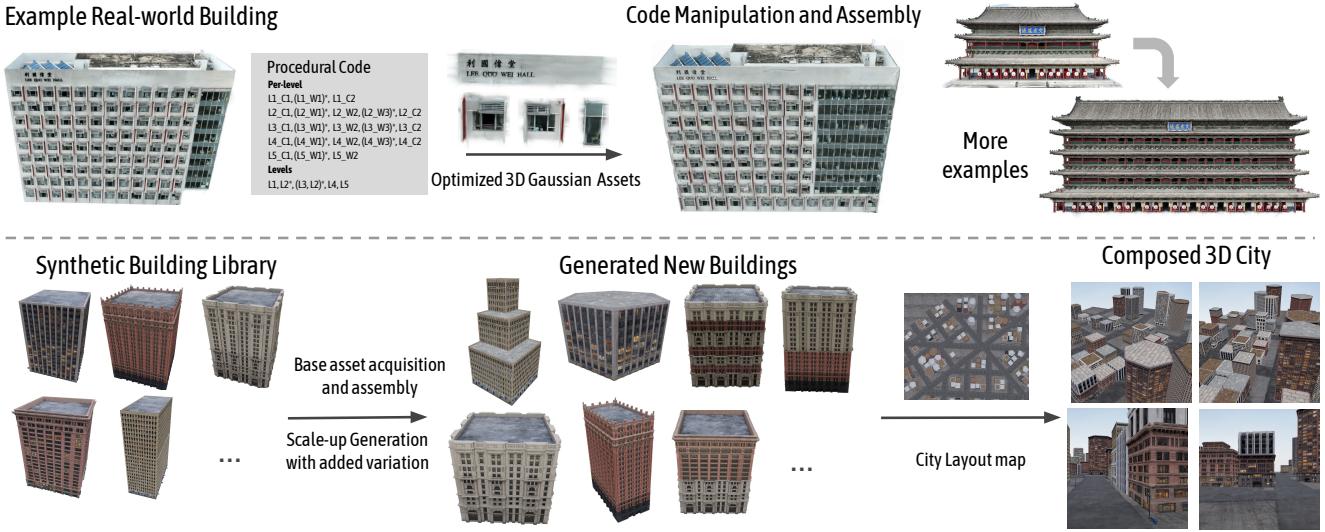


Figure 1. Architectural structures in urban environments often exhibit repetitive patterns, such as the arrangement of windows and doors in buildings. Our approach focuses on extracting 3D base assets from predefined layouts and multi-view captures of buildings. Utilizing 3D-GS for reconstruction, we incorporate procedural code constraints during optimization to decompose the building entity into distinct 3D assets. (1) The top row illustrates two real-world building examples, showcasing their appearances before and after editing. (2) The bottom row demonstrates the scaled-up city-level assembly achieved using UE's City Sample assets. ProcGS not only facilitates geometry editing but also enables the generation of new buildings by modifying or defining new procedural rules, offering a flexible, efficient, and scalable framework for city assembly with high-fidelity and precise control.

Abstract

Buildings are primary components of cities, often featuring repeated elements such as windows and doors. Traditional 3D building asset creation is labor-intensive and requires specialized skills to develop design rules. Recent generative models for building creation often overlook these patterns, leading to low visual fidelity and limited scalability. Drawing inspiration from procedural modeling techniques used in the gaming and visual effects industry, our method, Proc-GS, integrates procedural code into the 3D Gaussian Splatting (3D-GS) framework, leveraging their advantages in high-fidelity rendering and efficient asset management from both worlds. By manipulating procedural code, we can streamline this process and generate an infinite variety of buildings. This integration significantly reduces model size by utilizing shared foundational assets, enabling scal-

able generation with precise control over building assembly. We showcase the potential for expansive cityscape generation while maintaining high rendering fidelity and precise control on both real and synthetic cases. Project page: <https://city-super.github.io/procgs/>.

1. Introduction

High-quality 3D city assets are essential for virtual reality, video games, film production, and autonomous driving simulations. While recent advances leverage deep generative models [12, 27, 39, 44, 45] to scale up city generation process, the rendered visual fidelity and geometric accuracy of synthesized 3D city scenes remain unsatisfactory. Buildings, as pivotal elements in urban landscapes, pose a considerable challenge for generative models due to their in-

tricate geometries and diverse appearances. Nevertheless, industries have a long history using procedural generation to create high-quality, diverse 3D structures, from architectural models to virtual cities, especially in film and game development. Typically, this procedure involves creating a set of base assets, *e.g.* window, corner and wall, designing procedural rules and configurations, and assemble assets accordingly. In such way, artists can construct scenes in a scalable fashion, as demonstrated in City Sample [1] project. However, creating base assets is non-trivial and requires considerable human efforts to create intricate meshes and textures.

Recently, 3D Gaussian Splatting (3D-GS) [22] has garnered significant attention for its photo-realistic visual quality, and efficient training and rendering. Its explicit nature make this representation interactable and becoming popular among various applications like VR/AR, game development, content creation etc. It is therefore attempting to adopt this representation in city generation tasks. However, 3DGS has primarily been used to model entire scenes, which are usually compositions of multiple objects or elements, from multi-view captures. Isolating a specific part is challenging. For instance, while it's straightforward to model an entire building from photos, focusing solely on each individual component of the building, becomes cumbersome and challenging.

We present *Proc-GS*, the first pipeline that enables procedural modeling with 3DGS. Our framework consists of two stages: (1) In the *Asset Acquisition* stage, we constrain the optimization of 3DGS by guiding it to follow a predefined layout. For example, when modeling a building with 3DGS, we start by generating its procedural code, either manually or using an off-the-shelf segmentation model. This code is used to initialize a set of Gaussians for each base asset of the building. These asset-specific Gaussians are then assembled according to the procedural code, and we optimize the assembled Gaussians as a whole using rendering loss. Figure 3 illustrates an example. Notably, repeated base assets will be updated synchronously; to capture appearance various and subtle change in geometry, we additionally learn a variance code for each asset. (2) In the *Asset Assembly* stage, we use procedural code to manipulate base assets, generating buildings with diverse geometric structures and photorealistic appearances. We demonstrate that these newly created architectures can be integrated with Houdini [2], allowing for highly scalable scene composition with intuitive controls.

To showcase the capabilities of *Proc-GS*, we curated the *MatrixBuilding* dataset from the City Sample [1], which contains multi-view images and procedural codes for 17 iconic buildings. Our *Proc-GS* approach enables flexible geometry editing and the creation of new structures by combining assets from different buildings, allowing users to

generate vast, customized virtual cities. We also migrate *Proc-GS* to real-world buildings, and enable the conversion from actual structures into virtual assets, supporting scalable, photo-realistic city generation that benefits games, autonomous driving, and embodied AI etc. Experiments demonstrate that *Proc-GS* outperforms previous city generation methods in both rendering and geometry quality.

In summary, our contributions are follows:

- *MatrixBuilding Dataset*: A collection of dense multi-view images paired with procedural code for 17 iconic buildings, capturing high-resolution details and diverse architectural styles.
- *Proc-GS*: The first framework that integrates procedural modeling with 3D-GS to accelerate 3D building asset creation, and extraction from the real world scenes. Our method enhances infinite city generation with high flexibility and photo-realistic visual quality.

2. Related Work

2.1. Advancements in Neural Rendering

Neural rendering techniques, utilizing implicit representations for 3D modeling, have revolutionized novel view synthesis with photo-realistic rendering qualities. Recent advances fall into two broad categories: 1) differentiable volume rendering and 2) rasterization-based methods. The most representative work of the former is NeRF [31], which encodes the scene into the weights of Multi-Layer Perceptrons (MLPs). Despite its extraordinary ability to handle view-dependent appearance, the lack of explicit geometry structure hinders easy editing and physical interactions, making tasks like object insertion, deletion, and replacement tedious [17, 41]. Additionally, the slow training and inference speed of NeRF and its variants [4, 47] limits their practicality for large-scale generation. Even with more advanced backbones [32], rendering efficiency and computational costs still lag behind traditional rasterization methods, limiting their real-time application potential. In contrast, 3D Gaussian Splatting [22], which projects 3D Gaussians to a 2D image plane via rasterization, achieves state-of-the-art rendering quality. The explicit nature of 3D Gaussians has enabled a variety of applications, including 3D generation [40], physical simulation [46], and editing [14].

Inspired by the efficiency and high fidelity of 3D-GS, this work explores their potential for scalable generation of 3D scenes through procedural code and asset construction.

2.2. Advances in City Generation

Recent advances in city generation have produced several notable approaches for creating complete urban environments. They could be categorized to three types: (1) whole city generation, (2) city layout generation and (3) city-view video generation. InfiniCity [27] first introduced infinite-

scale synthesis through a three-module system with octree-based voxel representation, followed by CityDreamer [45] which proposed a compositional approach separating building instances from background elements. CityGen [12] further advanced layout generation using outpainting and diffusion models, while recent methods like UrbanWorld [39] pioneered a comprehensive pipeline combining diffusion-based rendering with multimodal language models, and GaussianCity [44] adapted 3D Gaussian splatting with BEV-Point representation for efficient large-scale rendering. In addition, many methods contribute to the task of city layout generation by introducing multi-modal controllable generation [50], a three-stage learning-based framework [13], graph-based modeling [18] and a comprehensive dataset [25]. Finally, video diffusion models [11] is also exploited for generating consistent city views.

Such generation-based methods commonly rely on generative priors to produce city views, yet their perceptual quality and 3D geometries could not be ensured. Our method pioneers the usage of procedural modeling and 3D-GS assets for better visual quality.

2.3. Procedural Modeling as Scalable Generators

Procedural generation involves creating a vast variety of assets using generalized rules and simulators. This technique has garnered significant interest in the computer vision and graphics community due to its scalability and adaptability. It is extensively used for creating virtual environments [9], urban areas [7, 30, 34, 42], and natural landscapes [23, 35, 38]. Additionally, procedural methods are employed for generating structured objects [24, 28, 30, 33] and textures [15, 19]. Procedural generation functions as a powerful data simulator, particularly valuable when obtaining or generating high-quality real data is challenging. Traditional rule-based procedural generators are integrated into popular 3D modeling software such as Blender, Houdini, and Unreal Engine, thereby streamlining the creation workflow for artists. At its core, procedural modeling represents world-building through concise mathematical rules, allowing for complex and varied structures to be efficiently created and manipulated. On the other hand, inverse procedural modeling addresses the challenge of inferring procedural rules from input data, either from 2D images [16, 33] or 3D models [10, 29]. This approach enables the extraction of procedural representations from existing assets, facilitating their integration into procedural workflows.

Inspired by these advancements, we aim to introduce procedural properties into 3D Gaussian Splatting (3D-GS). By leveraging the advantages of procedural generation and 3D-GS, we can enhance the flexibility and scalability of 3D scene generation, enabling the creation of high-fidelity, expansive virtual environments.

3. Procedural Modeling with 3D Gaussians

Our *Proc-GS* consists of two stages: (1) *Asset Acquisition* uses procedural code during the training process of 3D Gaussian Splatting (3D-GS) [22] to acquire base assets; (2) *Asset Assembly* manipulates the procedural codes to generate diverse buildings and assemble a 3D City. In the following sections, Sec.3.1 introduces the basics of 3D-GS, Sec.3.2 introduces the definition of procedural code, the synthetic *MatrixBuilding* dataset and how to obtain procedural code from real world, Sec. 3.3 and Sec. 3.4 separately introduces the *Asset Acquisition* and *Asset assembly*.

3.1. Preliminaries: 3D Gaussian Splatting

Unlike the implicit neural fields of NeRF [31], 3D-GS [22] represents 3D scenes with explicit anisotropic 3D Gaussians, maintaining differentiability while enabling efficient tile-based rasterization. Initialized from a set of sparse point cloud, each 3D Gaussian i is assigned with the learnable parameters $\{\mu_i, R_i, S_i, \alpha_i, \mathcal{C}_i\}$. For a given 3D point x within the scene,

$$G_i(x) = e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)}, \quad \Sigma_i = R_i S_i S_i^T R_i^T, \quad (1)$$

where μ_i represents the center of the Gaussian, R_i and S_i together define the covariance matrix Σ_i of the Gaussian. The opacity is denoted by α_i , which is multiplied by the Gaussian function $G_i(x)$ to determine the contribution of the final rendered output during the blending process. \mathcal{C}_i represents the spherical harmonic coefficients used to obtain the view-dependent color.

3D-GS uses tile-based rasterization for efficient scene rendering, projecting 3D Gaussians G onto the image plane as 2D Gaussians G' . The rasterizer sorts these 2D Gaussians and applies α -blending:

$$G'(x') = \sum_{i \in N} G_i(x') \alpha_i, \quad (2)$$

where x' is the pixel position, N is the number of 2D Gaussians, and d_i is the view direction to the center of G_i . This differentiable rasterizer allows direct optimization of 3D Gaussian parameters under training view supervision. 3D-GS excels in creating high-quality 3D assets efficiently, with its discrete structure that simplifies editing and cross-scene transfer.

$$C(x') = \sum_{i \in N} \text{SH}(d_i; \mathcal{C}_i) \sigma_i \prod_{j=1}^{i-1} (1 - \sigma_j), \quad \sigma_i = \alpha_i G'_i(x') \quad (3)$$

3.2. Procedural Code Definition

Procedural modeling is widely used for building generation in game scenes, such as the City Sample in UE5 [1, 3], by leveraging buildings' structured nature and repetitive assets.

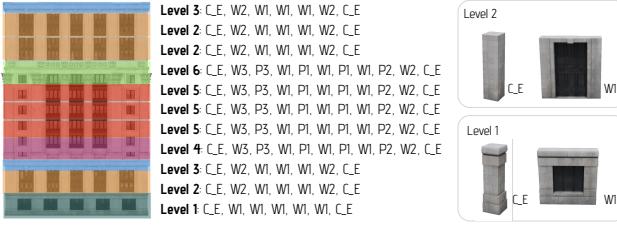


Figure 2. **Example Building Procedural Code from City Sample [1].** Different levels are distinguished by colors, each represented by a string of characters indicating the instantiation of assets: *C.E* (external corner), *P** (pillar), and *W** (window). Base assets such as *C.E* and *W1* for Levels 1 and 2, shown on the right, are manually created by artists.

Buildings are hierarchically decomposed into *levels*, where identical layers may repeat, as shown by Levels 2, 3, and 5 in Figure 2. Each level consists of base assets like windows, corners, and pillars, with identical assets recurring within levels, such as *C.E* and *W1* in Level 1. A building can thus be represented by a procedural code string and a set of base assets, where base assets are crafted manually by artists.

In this paper, we propose an alternative approach to extract base assets from multi-view images. To simplify the problem, we first assume the ground truth procedural code is provided. We created the *MatrixBuilding* dataset based on 17 buildings from the City Sample [1], crafted by artists in Maya to emulate architectural styles from cities like New York, Chicago, and San Francisco. Each scene includes dense multi-view images, ground-truth camera poses, and ground-truth procedural codes. More details are provided in the Appendix A. Our framework can operate on synthetic worlds and is also practical for real-world scenarios. We discovered an efficient approach to obtain procedural code from real-world scenes. We first train 2D-GS [20] to obtain geometrically accurate point clouds and mesh. Subsequently, building facades are automatically estimated using the method proposed in [48]. For each face, we render an image directly facing the building facade and annotate the procedural code on the 2D image. Then the 2D procedural code are projected onto the mesh to obtain the 3D procedural code. Please refer to the Appendix B for more details. Once the procedural code is obtained, our algorithm works almost identically for both real-world and synthetic scenes.

3.3. Asset Acquisition

In the gaming and animation industry, base assets are usually manually created by artists and assembled using either human-defined or heuristically generated procedural code. Our goal is to *extract* these 3D base assets automatically during the training process of 3D-GS [22]. To achieve this, we assume procedural code is available, whether it's the ground truth code from the *MatrixBuilding* dataset or estimated code from real-world scenes.

In addition to the procedural code, as shown in Fig. 2, we obtain for each base asset:

- the size of the asset’s bounding box, (x_e, y_e, z_e) ;
- the pivot location, (x_c, y_c, z_c) in its local coordinates;
- the set of transformations for K instantiations in the world coordinate system $\{[R_1, T_1, S_1], [R_2, T_2, S_2], \dots, [R_K, T_K, S_K]\}$, where $T \in \mathbb{R}^{3 \times 1}$ is the translation vector, $R \in \mathbb{R}^{3 \times 3}$ is the rotation matrix and $S \in \mathbb{R}^{3 \times 1}$ is the scale factor.

Gaussian Initialization. We initialize the pivot of each base asset at the origin of the world coordinate system, where the pivot is the origin of the asset’s local coordinate system. The bounding box of the i -th base asset in the world coordinate system is represented as $(x_{min}^i, y_{min}^i, z_{min}^i, x_{max}^i, y_{max}^i, z_{max}^i)$, where $x_{min}^i = x_c^i - \frac{x_e^i}{2}$ and $x_{max}^i = x_c^i + \frac{x_e^i}{2}$. The same calculation applies to the other two dimensions. The operations for synthetic and real-world scenes have subtle differences.

In synthetic scenes, for each building composed of a set of base assets \mathcal{M} , we initialize N points. For the i -th asset, N^i points are randomly initialized within its bounding box, determined by the ratio of the asset’s bounding box volume to the total volume of all assets, as shown in Equation 4:

$$N^i = N * \frac{V^i}{\sum_{j \in \mathcal{M}} V^j}, \quad V^i = x_e^i \times y_e^i \times z_e^i. \quad (4)$$

No matter in real-world or virtual world, each instantiation of the i -th base asset will have minor differences in appearance and geometry, so we initialize a variance asset for each instantiation of the i -th base asset. For the j -th instantiation, we first randomly initialize N^i points within the bounding box of i -th base asset. Then we update the center μ , rotation R and scale S of the 3D Gaussians in this variance asset, as shown in Equation 5.

For real-world scenes, given the i -th base asset with K instantiations, we apply transformations to map its bounding box to obtain the bounding boxes of all K instantiations. For each instantiation, we filter SfM points within its bounding box for variance initialization. These point clouds are then transformed back to the world coordinate origin using inverse transformations of the i -th base asset and concatenated. Finally, we uniformly downsample the resulting point cloud by a factor of K to initialize the i -th base asset.

Rendering with Procedural Code. Figure 3 (1) illustrates our rendering pipeline. First, we assemble the base assets according to the procedural code. The i -th base asset has a set of 3D Gaussians \mathcal{A}_i and transformations $\{[R_1^i, T_1^i, S_1^i], [R_2^i, T_2^i, S_2^i], \dots, [R_j^i, T_j^i, S_j^i], \dots, [R_K^i, T_K^i, S_K^i]\}$. For the j -th instantiation of the i -th base asset, the 3D Gaussian properties remain the same except for the center μ , rotation R and scale S , which are updated as:

$$\mu' = R_j^i \cdot S_j^i \cdot \mu + T_j^i, \quad R' = R_j^i \cdot R, \quad S' = S_j^i \cdot S. \quad (5)$$

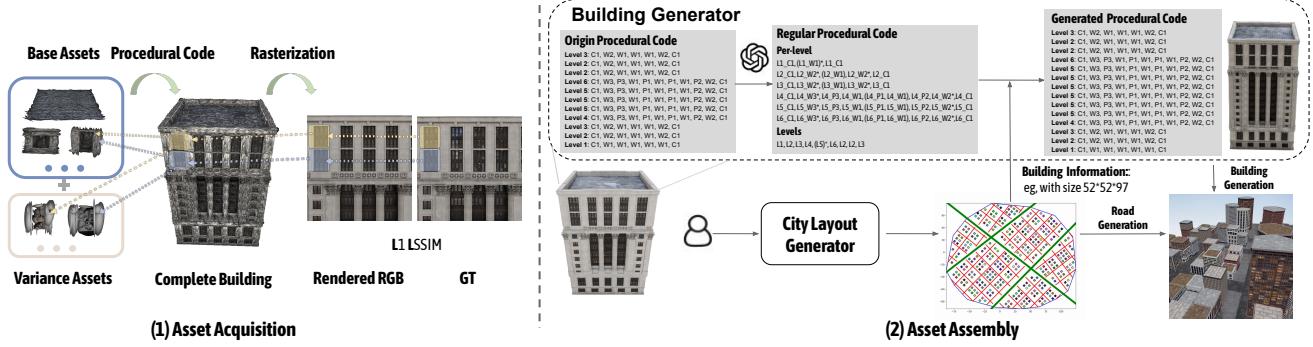


Figure 3. Overview of ProcGS. Our pipeline consists of two stages: (1) **Asset Acquisition**: We acquire the base assets in the training process of the 3D-GS. These assets are then assembled according to procedural code and add variance assets to create a complete building, which is used for novel view synthesis with Gaussian Splatting. (2) **Asset Assembly**: We use the *building generator* and *city layout generator* to assemble these base assets into a vivid 3D city. Users provide basic urban spatial data (purple city boundary and green primary road) to city layout generator to automatically predict other roads and building layouts. The building generator then assembles base assets into complete buildings using procedural code and predicted building parameters.

After instantiating all the base assets according to the procedural code and adding the variance assets, the complete building is formed, as shown in Figure 3 (1). The 3D Gaussians of the complete building are then fed into the rasterizer to render the image, supervised by the training views. A base asset may appear multiple times within the same image. During optimization, gradients from these repeated instances are backpropagated to the shared base asset and respective variance assets, refining the Gaussian parameters.

Bbox Adaptive Control of Gaussians. The internal ordering of Gaussians can be chaotic, often leading to good rendering results but disordered boundaries for base assets, complicating subsequent editing and generation (Figure 6). To address this, we enhance the original 3D-GS [22] with a *Bbox Adaptive Clamp* operation (Figure 4) in addition to densification and pruning. For each base asset, the *Bbox Adaptive Clamp* operation involves: (1) Clamp Scale: Using a slightly larger ‘soft’ bounding box to avoid over-clamping. If a Gaussian exceeds the soft box boundaries, its scale is halved. (2) Clamp Position: Pulling the centers of Gaussians exceeding the bounding box back to its edge. This operation is performed every 100 iterations for both base and variance assets to maintain ordered boundaries and facilitate efficient extraction and manipulation.

Loss Functions. We optimize the learnable Gaussians’ parameters with respect to the \mathcal{L}_1 loss over rendered pixel colors and SSIM term [43] $\mathcal{L}_{\text{SSIM}}$. The total supervision is given by $\mathcal{L} = \mathcal{L}_1 + \lambda_{\text{SSIM}} \mathcal{L}_{\text{SSIM}}$.

3.4. Asset Assembly

After extracting base assets from multi-view images, we can manipulate procedural code to generate new buildings. Combined with a rule-based 2D city layout generator, we can assemble complete 3D city scenes. As shown in Figure 3 (2), the assembly process consists of a building gen-

erator and a city layout generator.

Building Generator For architectural generation, we first analyze the arrangement patterns of base assets within each floor and between floors in the original building, converting them into regular procedural code through GPT-4o [21] with several examples provided in the prompt. The detailed process is explained in the Appendix C. In the regular procedural code, repeatable and scalable combinations both within and between floors are specified. Concretely, combinations within () are designed as repeatable elements, while assets marked with * are scalable to fit the size of the building. Subsequently, we can place assets according to the procedural code and specified building size (e.g., length, width and height) to generate new buildings with varying arrangements and sizes. Notably, during the building generation process, each base asset is randomly assigned a corresponding variance asset to enhance diversity and realism. Besides, we could also create new buildings from base assets extracted from different architectural sources.

City Layout Generator For city generation, users first select boundary points of the city map (purple boundary) and primary roads’ endpoints (green lines), as shown in Figure 3 (2). Subsequently, we partition the city into several connected blocks based on the primary roads, with each block being randomly assigned regional characteristics (e.g., distribution of building sizes). Following this, we generate perpendicular secondary roads and determine building positions, topological structures, and sizes according to predefined rules. Finally, we randomly place decorative elements along the roads, such as street lamps, garbage bins, and mailboxes, which are also collected by the 3D-GS.

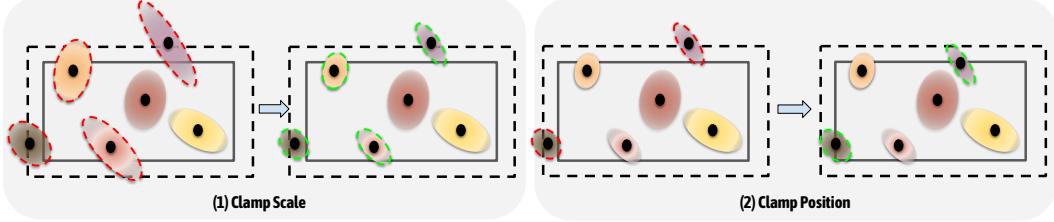


Figure 4. **Bbox Adaptive Clamp.** (1) Clamping Scale: if a Gaussian exceeds the soft bounding box, the scale of this Gaussian is reduced by half. (2) Clamping Position: repositions Gaussians that exceed the bounding box, realigning them within the bounding box.

4. Experiments

4.1. Experimental Setup

Dataset. We collected the *MatrixBuilding* dataset, featuring dense multi-view images of 17 buildings and their corresponding ground truth procedural codes from the City Sample [1]. Following MatrixCity [26], we disabled motion blur and used anti-aliasing during rendering to achieve the highest possible image quality. Details for each building and camera capture trajectory are provided in the Appendix A. We also validated our method on three real-world scenes captured by drones.

Implementations. We selected 3D-GS [22] as our primary baseline due to its state-of-the-art performance in novel view synthesis. Both 3D-GS and our proposed method were trained for 30k iterations and densified until 15k iterations. The number of gaussians initialized for synthetic data, N , in our method is set to 10k, as illustrated in Equation 4. 3D-GS is initialized with the building assembled by the randomly initialized base assets according to the procedural codes. The soft bounding box is expanded by 20cm beyond the bounding box. The loss weight λ_{SSIM} is set to 0.2. Our model is trained on single RTX 3090 GPU.

Metrics. For novel view synthesis, we report widely adopted metrics: PSNR, SSIM [43], and LPIPS [49]. Additionally, we report the number of Gaussians to evaluate model compactness. The metrics are averaged across all scenes for quantitative comparison. For city generation, we report **Depth Error** (DE) and **Camera Error** (CE) following EG3D [6] and CityDreamer [45] to evaluate the 3D scene geometry and consistency. For DE, we utilize a pre-trained model [36] to estimate the depth maps from the rendered images and calculate the ℓ_2 distance between the normalized estimated depth and rendered depth. For CE, we first render images using hemispherically sampled camera poses, then estimate these poses using COLMAP [37]. The camera error is computed as a scale-invariant ℓ_2 loss between the estimated and ground truth poses.

4.2. Comparisons with 3D-GS

In Table 1 and Figure 5, we compare our Proc-GS with the original 3D-GS on both synthetic and real-world scenes.

Data Type	Method	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	#GS (k) \downarrow
Synthetic	3D-GS	27.54	0.910	0.108	1,238
	Proc-GS (ours)	27.68	0.917	0.102	291
Real	3D-GS	27.38	0.858	0.192	500
	Proc-GS (ours)	27.19	0.853	0.196	384

Table 1. **Quantitative comparisons.** Proc-GS achieves similar perceptual quality while requiring less memory and providing flexible control capabilities, demonstrating the effectiveness of integrating procedural code with 3D-GS for building scenes.

Views	3D-GS [22]			Proc-GS		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
24	16.93	0.542	0.410	19.70	0.682	0.294
47	20.65	0.688	0.283	23.11	0.795	0.196
469	27.54	0.910	0.108	27.68	0.917	0.102

Table 2. **Sparse view quantitative results.** Proc-GS is robust to the view elimination and outperforms the baseline by a large margin under sparse-view setting on *MatrixBuilding* Dataset, showing the fitness to optimization.

For synthetic data, Proc-GS achieves comparable novel view synthesis quality while significantly reducing the model size by a factor of 4, demonstrating the effectiveness of our approach. For real-world scenes, due to their complex appearance, geometry, and lack of ground-truth procedural code, the optimization becomes more challenging and requires more Gaussians for the variance assets. Although the compression rate is lower than in synthetic cases, our model still maintains a smaller size compared to 3D-GS. More importantly, we gain flexible control capability with only a slight decrease in accuracy, which is barely noticeable in the qualitative results (Figure 5).

Another significant strength of our Proc-GS is its ability to reconstruct from sparse view inputs, as depicted in Table 2. Using the dense view benchmark of 3D-GS, our Proc-GS delivers similar results with 3D-GS on *MatrixBuilding* dataset. However, 3D-GS with fewer views delivers significantly inferior results compared to the dense view scenario. In contrast, our Proc-GS maintains robustness against the reduction in the number of training views since the repeated occurrence of base assets naturally serves as a form of data augmentation. More qualitative comparisons are

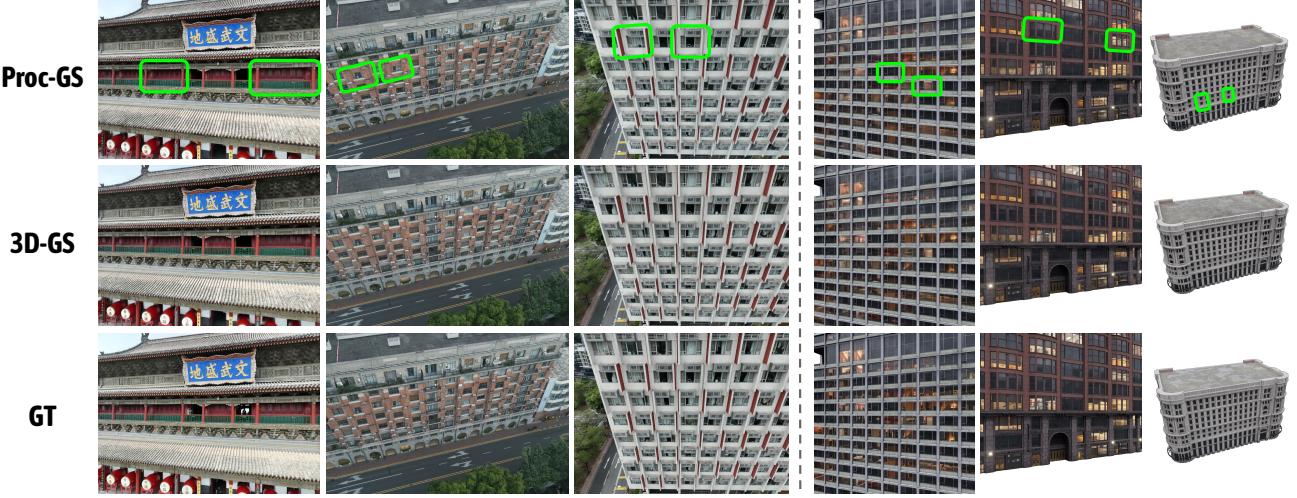


Figure 5. **Qualitative results.** The left section shows results from three real-world scenes, while the right section presents results from the *MatrixBuilding* dataset. Proc-GS achieves rendering quality comparable to 3D-GS. Green boxes in each image highlight pairs of instantiations that share the same base assets, illustrating our method’s capability to effectively model variations in geometry and appearance.

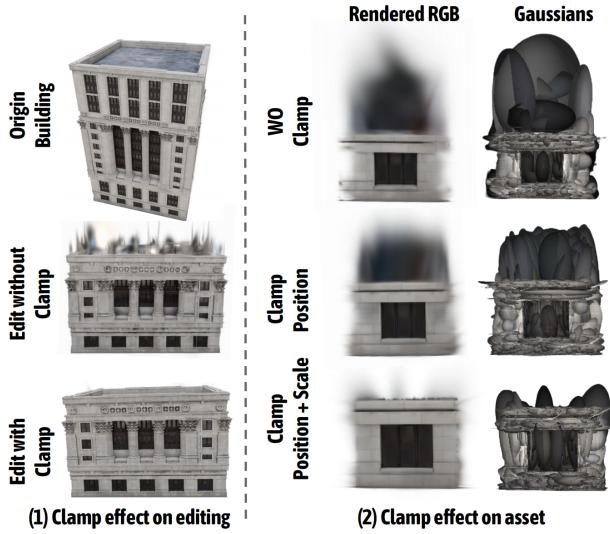


Figure 6. (1) **Clamp effect on editing.** Without clamp, the boundary area of edited scene is intensively corrupted by artifacts, making it impractical to create a new building with these assets; (2) **Clamp effect on asset.** We ablate effects of the clamp operation and demonstrate the effectiveness of both strategies.

provided in the Appendix D.1. The robustness of Proc-GS with sparse views shows great potential for real-world applications as the data collection process is cumbersome and there will be many scenes that very limited views of data is accessible.

4.3. Ablation Studies

Clamp strategies. Building components are seamlessly connected. But during component decoupling, Gaussian

	PC	BC	VR	PSNR↑	SSIM↑	LPIPS↓	#GS (k)↓
1				27.54	0.910	0.108	1238
2	✓			25.54	0.903	0.123	87
3	✓	✓		24.40	0.892	0.132	87
4	✓	✓	✓	27.68	0.917	0.102	291

Table 3. **Quantitative ablations** on Procedural Code (PC), Box Adaptive Clamp (BC), and Variance (VR) using the *MatrixBuilding* dataset. Row 1 indicates the vanilla 3D-GS baseline.

kernels often extend beyond asset boundaries. This overlap complicates asset combination and building editing, as demonstrated in Figure 6 (1). To address this issue, we developed clamp strategies as illustrated in Figure 4. Figure 6 (2) shows a qualitative evaluation of our clamp operations in Proc-GS. The implementation of these two clamp strategies results in significantly cleaner asset boundaries.

Variance assets. However, as shown in Table 3, we find that integrating with the procedural code and adding the clamp operations both leads to worse rendering quality. This is because the instantiations of base assets are not completely identical. When different instantiations do not affect each other at all, the ability to model diversity is minimized. Therefore, we have added a variance asset to each instantiation, which allows us to achieve similar performance as 3D-GS. In the *Asset Assembly* process, we randomly assign variance assets to each instantiation to enhance the diversity of generated buildings.

4.4. Results on City Generation

Our Proc-GS can extract base assets in the training process of 3D-GS based on procedural codes. By manipulating procedural codes, we can flexibly edit the building geom-

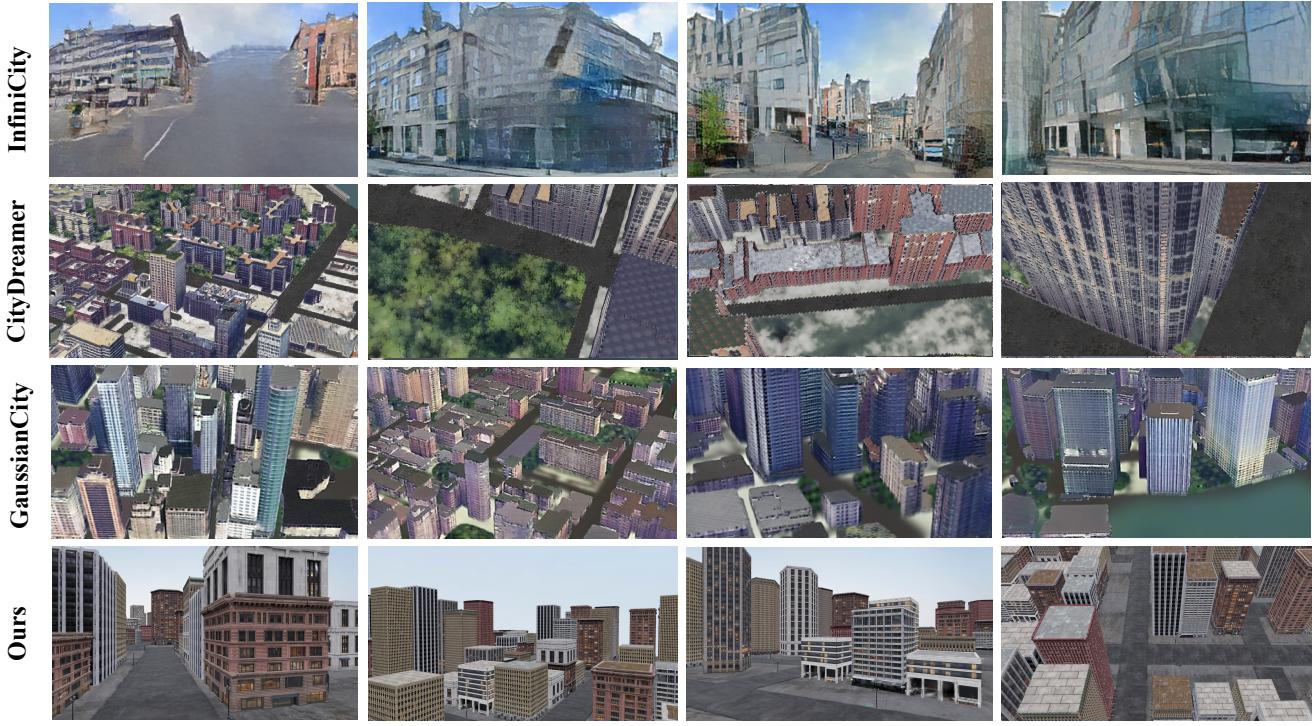


Figure 7. **Qualitative comparisons of city generation results.** Our Proc-GS framework learns base assets during the training process of 3D-GS using procedural codes, which are then manipulated to assemble these assets into a cohesive 3D city. Compared to other generation-based methods, Proc-GS demonstrates superior visual quality in both aerial and street-level views, especially in architectural details. We recommend zoom-in for detailed inspection.

Method	Camera Error↓	Depth Error↓
Pers. Nature [5]	86.371	0.109
SceneDreamer [8]	0.186	0.216
CityDreamer [45]	0.060	0.096
GaussianCity [44]	0.057	0.090
Ours	0.049	0.032

Table 4. **Quantitative comparison** of Depth and Camera Error. Our method outperforms existing approaches.

try, generate new building using cross-scene base assets and assemble these buildings into city scenes as shown in Figure 1. More qualitative results are provided in the Appendix D.2. In Table 4, we compare our assembled city with other baseline methods quantitatively. Our approach achieves better scores on CE and DE metrics, indicating superior 3D consistency in the generated city scenes. We compare with other methods qualitatively in Figure 7. We demonstrate enhanced visual quality, more stable generation, and greater flexibility in control.

5. Limitations and Potentials

While we have demonstrated the possibility to obtain high-quality base assets from real scenes, scaling up this process faces several challenges: 1) fully automating procedu-

ral code generation without any human intervention; 2) extracting high-quality base assets from sparse views or even single images to reduce data collection costs. Additionally, our current layout generation does not consider aesthetic principles or urban functionality. In the future, LLMs could replace rule-based systems to generate more practical and aesthetically pleasing urban layouts. This will allow us to collect a wealth of base assets from real scenes and automatically assemble them into virtual cities, enhancing practical applications that rely on photorealistic data, such as embodied AI and autonomous driving.

6. Conclusion

In this paper, we propose Proc-GS, a novel approach to efficiently craft high-quality base building assets by utilizing procedural code during the 3D Gaussian Splatting (3D-GS) training process. Proc-GS decomposes a complete Gaussian model of a building into base assets and a procedural code string. This decomposition enables flexible editing of building geometries and the creation of diverse structures by combining base assets from different scenes, thereby supporting the generation of extensive cityscapes. Proc-GS leverages the efficient rendering capabilities and the discrete structure of 3D-GS, and demonstrates its versatility on both synthetic and real-world scenarios.

References

- [1] <https://www.unrealengine.com/marketplace/product/city-sample>. 2, 3, 4, 6, 11
- [2] <https://www.sidefx.com/>. 2
- [3] <https://www.unrealengine.com/>. 3, 11
- [4] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, pages 5460–5469. IEEE, 2022. 2
- [5] Lucy Chai, Richard Tucker, Zhengqi Li, Phillip Isola, and Noah Snavely. Persistent nature: A generative model of unbounded 3d worlds. In *CVPR*, pages 20863–20874. IEEE, 2023. 8
- [6] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, pages 16102–16112. IEEE, 2022. 6
- [7] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. *ACM Trans. Graph.*, 27(3):103, 2008. 3
- [8] Zhaoxi Chen, Guangcong Wang, and Ziwei Liu. Scene-dreamer: Unbounded 3d scene generation from 2d image collections. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(12):15562–15576, 2023. 8
- [9] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. In *NeurIPS*, 2022. 3
- [10] Ilke Demir, Daniel G. Aliaga, and Bedrich Benes. Proceduralization for editing 3d architectural models. In *3DV*, pages 194–202. IEEE Computer Society, 2016. 3
- [11] Boyang Deng, Richard Tucker, Zhengqi Li, Leonidas J. Guibas, Noah Snavely, and Gordon Wetzstein. Streetscapes: Large-scale consistent street view generation using autoregressive video diffusion. In *SIGGRAPH (Conference Paper Track)*, page 27. ACM, 2024. 3
- [12] Jie Deng, Wenhao Chai, Jianshu Guo, Qixuan Huang, Wenhao Hu, Jenq-Neng Hwang, and Gaoang Wang. Citygen: Infinite and controllable 3d city layout generation. *arXiv preprint arXiv:2312.01508*, 2023. 1, 3
- [13] Jie Deng, Wenhao Chai, Junsheng Huang, Zhonghan Zhao, Qixuan Huang, Mingyan Gao, Jianshu Guo, Shengyu Hao, Wenhao Hu, Jenq-Neng Hwang, Xi Li, and Gaoang Wang. Citycraft: A real crafter for 3d city generation. *arXiv preprint arXiv:2406.04983*, 2024. 3
- [14] Jiemin Fang, Junjie Wang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians delicately with text instructions. *arXiv preprint arXiv:2311.16037*, 2023. 2
- [15] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanaprasam, Florian Golemo, Charles Herrmann, et al. Kubric: A scalable dataset generator. In *CVPR*, pages 3749–3761, 2022. 3
- [16] Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. Inverse procedural modeling of branching structures by inferring l-systems. *ACM Trans. Graph.*, 39(5):155:1–155:13, 2020. 3
- [17] Ayaan Haque, Matthew Tancik, Alexei A Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. *arXiv preprint arXiv:2303.12789*, 2023. 2
- [18] Liu He and Daniel G. Aliaga. COHO: context-sensitive city-scale hierarchical urban layout generation. *arXiv preprint arXiv:2407.11294*, 2024. 3
- [19] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly E. Rushmeier, and Valentin Deschaintre. Generating procedural materials from text or image prompts. In *SIGGRAPH (Conference Paper Track)*, pages 4:1–4:11. ACM, 2023. 3
- [20] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH (Conference Paper Track)*, page 32. ACM, 2024. 4, 11, 12
- [21] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 5, 11, 13
- [22] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139:1–139:14, 2023. 2, 3, 4, 5, 6, 11, 13
- [23] Samin Khan, Buu Phan, Rick Salay, and Krzysztof Czarnecki. Procsy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *CVPR Workshops*, pages 88–96. Computer Vision Foundation / IEEE, 2019. 3
- [24] Bosheng Li, Jonathan Klein, Dominik L. Michels, Bedrich Benes, Sören Pirk, and Wojtek Palubicki. Rhizomorph: The coordinated function of shoots and roots. *ACM Trans. Graph.*, 42(4):59:1–59:16, 2023. 3
- [25] Tao Li, Ruihang Li, Huangnan Zheng, Shanding Ye, Shijian Li, and Zhijie Pan. Robus: A multimodal dataset for controllable road networks and building layouts generation. *arXiv preprint arXiv:2407.07835*, 2024. 3
- [26] Yixuan Li, Lihan Jiang, Linning Xu, Yuanbo Xiangli, Zhenzhi Wang, Dahua Lin, and Bo Dai. Matrixcity: A large-scale city dataset for city-scale neural rendering and beyond. In *ICCV*, pages 3182–3192. IEEE, 2023. 6
- [27] Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang, and Sergey Tulyakov. Infiniticity: Infinite-scale city synthesis. *arXiv preprint arXiv:2301.09637*, 2023. 1, 2
- [28] Liane Makatura, Bohan Wang, Yi-Lu Chen, Bolei Deng, Chris Wojtan, Bernd Bickel, and Wojciech Matusik. Procedural metamaterials: A unified procedural graph for metamaterial design. *ACM Trans. Graph.*, 42(5):168:1–168:19, 2023. 3
- [29] Markus Mathias, Andelo Martinovic, Julien Weissenberg, and Luc Van Gool. Procedural 3d building reconstruction

- using shape grammars and detectors. In *3DIMPVT*, pages 304–311. IEEE Computer Society, 2011. 3
- [30] Paul Merrell. Example-based procedural modeling using graph grammars. *ACM Trans. Graph.*, 42(4):60:1–60:16, 2023. 3
- [31] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV (1)*, pages 405–421. Springer, 2020. 2, 3
- [32] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 2
- [33] Gen Nishida, Adrien Bousseau, and Daniel G. Aliaga. Procedural modeling of a building from a single image. *Comput. Graph. Forum*, 37(2):415–429, 2018. 3
- [34] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *SIGGRAPH*, pages 301–308. ACM, 2001. 3
- [35] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *CVPR*, pages 12630–12641, 2023. 3
- [36] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(3):1623–1637, 2022. 6
- [37] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, pages 4104–4113. IEEE Computer Society, 2016. 6
- [38] Hugo Schott, Axel Paris, Lucie Fournier, Eric Guérin, and Eric Galin. Large-scale terrain authoring through interactive erosion simulation. *ACM Trans. Graph.*, 42(5):162:1–162:15, 2023. 3
- [39] Yu Shang, Yuming Lin, Yu Zheng, Hangyu Fan, Jingtao Ding, Jie Feng, Jiansheng Chen, Li Tian, and Yong Li. Urbanworld: An urban world model for 3d city generation. *arXiv preprint arXiv:2407.11965*, 2024. 1, 3
- [40] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. *arXiv preprint arXiv:2309.16653*, 2023. 2
- [41] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Er-rui Ding, Jingdong Wang, and Gang Zeng. Delicate textured mesh recovery from nerf via adaptive surface refinement. *arXiv preprint arXiv:2303.02091*, 2023. 2
- [42] Apostolia Tsirikoglou, Joel Kronander, Magnus Wrenninge, and Jonas Unger. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. *arXiv preprint arXiv:1710.06270*, 2017. 3
- [43] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.*, 13(4):600–612, 2004. 5, 6
- [44] Haozhe Xie, Zhaoxi Chen, Fangzhou Hong, and Ziwei Liu. Gaussiancity: Generative gaussian splatting for unbounded 3d city generation. *arXiv preprint arXiv:2406.06526*, 2024. 1, 3, 8
- [45] Haozhe Xie, Zhaoxi Chen, Fangzhou Hong, and Ziwei Liu. Citydreamer: Compositional generative model of unbounded 3d cities. In *CVPR*, pages 9666–9675. IEEE, 2024. 1, 3, 6, 8
- [46] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198*, 2023. 2
- [47] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *CVPR*, pages 5438–5448, 2022. 2
- [48] Mulin Yu and Florent Lafarge. Finding good configurations of planar primitives in unorganized point clouds. In *CVPR*, pages 6367–6376, 2022. 4, 11, 12
- [49] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595. IEEE, 2018. 6
- [50] Shougao Zhang, Mengqi Zhou, Yuxi Wang, Chuanchen Luo, Rongyu Wang, Yiwei Li, Xucheng Yin, Zhaoxiang Zhang, and Junran Peng. Cityx: Controllable procedural content generation for unbounded 3d cities. *arXiv preprint arXiv:2407.17572*, 2024. 3

Proc-GS: Procedural Building Generation for City Assembly with 3D Gaussians

Appendix

A. More Dataset Details

Our *MatrixBuilding* Dataset consist of 17 buildings from the City Sample Project [1, 3], as shown in Figure 8 (a), which contains ground-truth procedural code and dense multi-view images. These buildings are created to mimic the building styles of Chicago, San Francisco, and New York. In Table 5, we also provide the number of building base assets and the total count of instantiated assets after assembling complete buildings according to procedural codes. The design of base assets combined with procedural codes significantly reduces the model size. Figure 8 (b) shows the dense camera capture trajectories. The ratio between training view and test view is about five to one.

B. Exploration of Real-World Scenes

Figure 9 illustrates our approach to extracting procedural code from real-world scenes. We begin by utilizing 2D-GS [20] to extract point clouds with good geometric structures from multi-view images, followed by automatically estimating building facades using the method of Yu et al. [48]. Buildings are composed of multiple facades. For each facade, we automatically render a maximally large image with comprehensive information directly facing the facade. Subsequently, we manually annotate 2D procedural code for each facade, and then project these annotations onto the 2D-GS mesh to derive the corresponding 3D procedural code. For each instantiation of the base assets, the range of the bounding box in the z-direction is calculated based on the current facade position and empirically set facade thickness. Automatically obtaining 2D procedural code could potentially be replaced by segmentation methods, which enables acquiring base assets from real-world scenes with minimal human intervention, beyond the initial data collection effort. We will conduct in-depth exploration of this direction in the future to enable large-scale collection of base assets from real-world scenes.

C. Prompt Example

To illustrate the process of obtaining regular procedural code from raw data, we include an example of the prompt used in our framework in Figure 10. The goal is to summarize repetitive and scalable structures within raw data and represent them concisely using regular expressions of procedural code. The raw data represents the configuration of a multi-layered building with modular patterns. For instance, a single row might include repetitive modules like *L1_W1*. Learning from one or more pairs of raw data and regu-

lar procedural codes, GPT-4o [21] could transforms raw data into a regularized procedural representation. We transform verbose raw data into a structured, succinct procedural summary that distills the input’s intrinsic regularities while maintaining human interpretability.

D. More Qualitative Results

D.1. Sparse View

Figure 11 shows the sparse view qualitative results. Unlike 3D-GS [22], which suffers from significant artifacts when reducing training views, Proc-GS exhibits remarkable robustness. The proposed design of shared base assets enables a natural data augmentation mechanism, where base assets are dynamically influenced by all instances throughout the training process. This characteristic significantly enhances our ability to extract base assets from sparse image sets, thereby substantially lowering the overall data collection expenses.

D.2. Building Editing

In Figure 12, we provide three building editing results from the real-world scene. We further showcase an intriguing building editing demo, where by manipulating variance assets, we precisely spelled out our method’s name on the building facade, thereby illustrating the remarkable controllability of our approach.

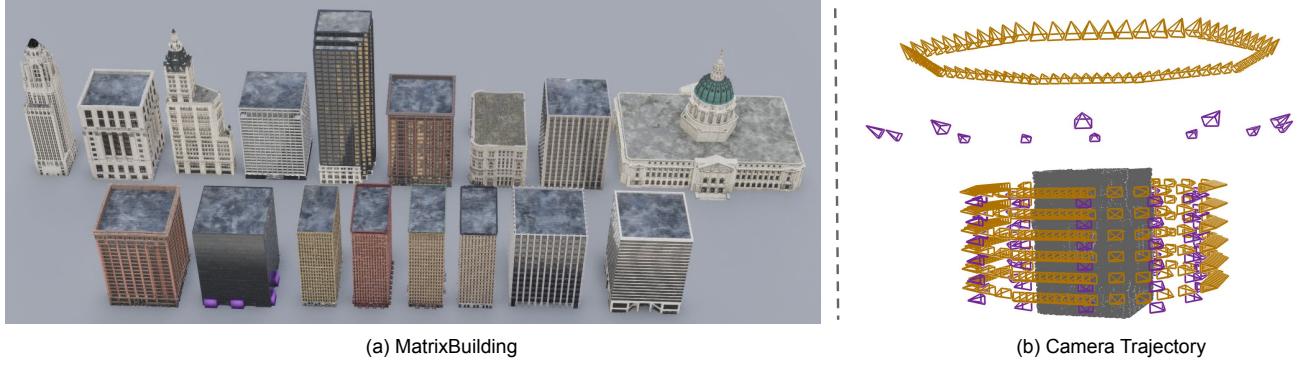


Figure 8. **Dataset Overview.** (a) Overview of the 17 buildings in our proposed *MatrixBuilding* dataset (b) Yellow cameras represent training views and purple cameras represent test views. The proportion of training views to test views is about 5:1.

Building	CHB	CHD	CHE	CHF	CHG	CHH	CHI	CHJ	NYAA	NYAB	NYAE	NYAF	NYG	SFA	SFB	SFD	SFE
# Base Assets	90	30	90	32	24	19	43	8	17	24	25	37	56	54	81	12	20
# Total Assets	1559	345	1645	1170	617	1585	697	2409	1869	1920	1929	2831	438	295	821	729	1405

Table 5. **Base Assets Statistics.** *CH** means a building of Chicago. *SF** means a building of San Francisco. *NY** means a building of New York. # Total Assets means the total count of instantiated assets after assembling complete buildings according to the procedural codes

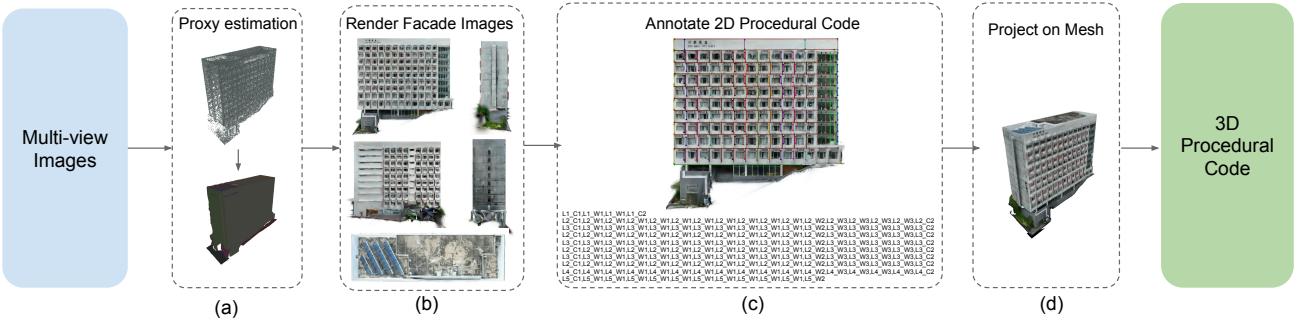


Figure 9. **Extracting Procedural Code from Real-World Scenes.** (a) We extract point clouds with good geometric structures from multi-view images using 2D-GS [20]. Then we use the method [48] to automatically estimate the building facade. (b) For each facade, automatically render a maximally large image with comprehensive information directly facing the facade. (c) Manually annotate 2D procedural code for each facade. (d) Project the 2D procedural code onto the 2D-GS mesh to derive the corresponding 3D procedural code.

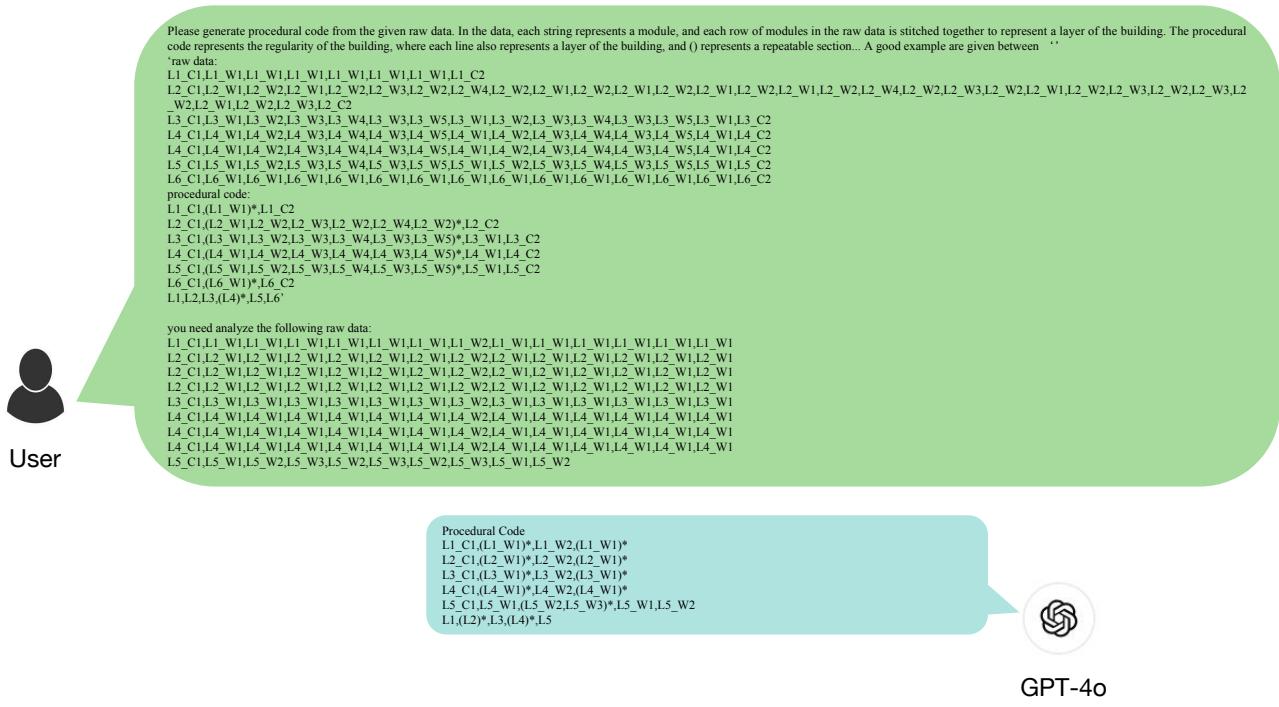


Figure 10. An example of the prompt used to obtain regular procedural code. GPT-4o [21] takes the raw data, one or more examples as well as descriptions of procedural code as input and summarizes the regular procedural code as output.

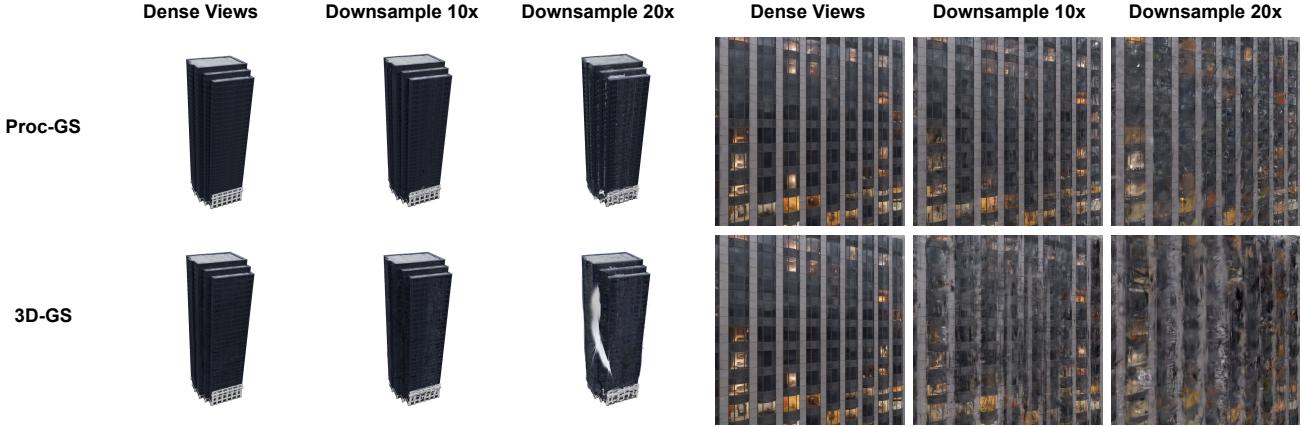


Figure 11. Sparse View qualitative results. Proc-GS demonstrates significant robustness when reduces the number of training views, in contrast to 3D-GS [22], which exhibits a pronounced susceptibility to numerous artifacts under similar conditions. This difference is attributed to the superior data efficiency inherent in our procedural code design, affirming its effectiveness in optimizing performance even with limited data inputs.

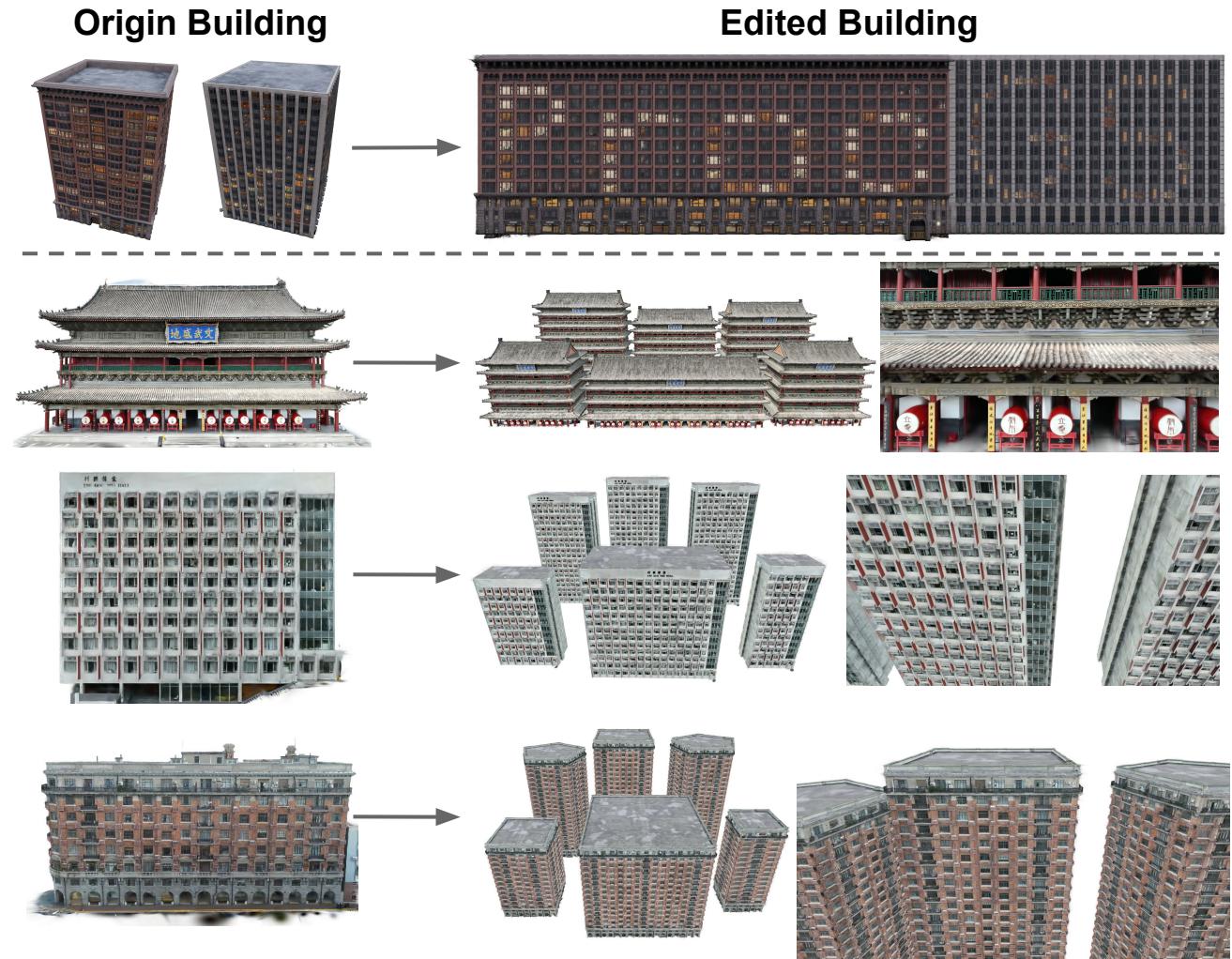


Figure 12. **Building Editing.** (1) The upper part shows synthetic data results, where we arranged variance assets to spell our method’s name, emphasizing its high controllability. (2) The lower part presents three editing results from the real-world scene.