Design and Implementation of a Low-Threshold Automatic Virtual Scene Generation Framework Based on Pix2Pix and Unity

Liangliang Jin
School of Xi'an Technology and Business College
Xi'an, China
E-mail: jinliangliang6@163.com

***Abstract*—To address the issues of low efficiency in manual modeling of virtual scenes and the high engineering threshold of existing AI generation technologies, an offline automatic virtual scene generation framework based on Generative Adversarial Networks (GAN) and the Unity engine is proposed. With the Pix2Pix conditional generation model as its core, this framework enables the automatic generation of indoor scene layouts. It completes the reconstruction from layouts to 3D scenes through the Unity parsing-generation module and adopts an "offline file collaboration" mode to reduce the cross-domain technical coupling. Experiments are conducted based on the SUNCG indoor dataset. The average Intersection over Union (IoU) between the layouts generated by the trained Pix2Pix model and the real layouts reaches 0.78. In a general PC environment, the Unity scene reconstruction module takes no more than 1.2 seconds to generate a 10×10m indoor scene, which improves the efficiency by over 99% compared with manual modeling. User experience tests show that this framework has a low operation threshold, and the generated scenes receive an average score of 4.1/5 in terms of layout rationality and practicality, which can meet the rapid development needs of scenarios such as game prototype design and educational virtual simulation.**

Keywords-Virtual Scene Generation; Generative Adversarial Network; Pix2Pix; Unity Engine; Offline Collaboration; Procedural Content Generation

I. INTRODUCTION

A. Research Background

With the rapid development of fields such as digital twins, game development, and virtual education, the demand for high-quality virtual scenes has experienced explosive growth [1]. Traditional virtual scene construction relies on manual modeling (e.g., 3ds Max, Blender), which has problems such as low efficiency, high cost, and poor consistency. For example, manual modeling of a moderately complex indoor scene (such as a classroom or office) takes 20-40 working hours, making it difficult to meet the needs of rapid iteration [2].

Procedural Content Generation (PCG) technology provides ideas for solving this problem. Among them, Generative Adversarial Networks (GAN) have demonstrated advantages in scene layout, texture, and 3D asset generation due to their strong "data-driven generation" capability [3]. For instance, the Pix2Pix model can realize the generation from semantic sketches to scene layouts through "image-to-image" mapping [4], and StyleGAN3 can generate diverse 3D object assets [5]. However, existing studies mainly focus on optimizing the generation accuracy of GAN models and lack engineering collaboration solutions with game engines (e.g., Unity). On the one hand, GAN model training requires a deep learning technology stack (Python/PyTorch), while Unity development relies on C# and the engine ecosystem, resulting in high cross-domain technical coupling. On the other hand, most collaboration solutions adopt "real-time data communication" (e.g., Socket, REST API), which requires solving problems such as multi-thread synchronization and delay control, posing a high threshold for small and medium-sized teams or non-professional developers [6].

B. Research Objectives and Significance

This paper aims to design a low-threshold and

highly implementable automatic virtual scene generation framework, with the core objectives as follows:

- Construct a full-process solution of "GAN layout generation → Unity scene reconstruction" and adopt an "offline file collaboration" mode to avoid the technical complexity of real-time communication;
- Reduce the technical threshold of model training and engine integration based on the mature Pix2Pix model and a step-by-step implementation strategy (from static parsing to AI generation);
- Verify the effectiveness of the framework in terms of generation quality, efficiency, and user experience through experiments, and provide a low-cost solution for teams with limited resources.

The research significance of this paper lies in filling the gap between "GAN generation technology" and "Unity engineering implementation", decomposing complex cross-domain tasks into modular steps that can be implemented step by step. This enables non-professional developers to quickly master the automatic virtual scene generation technology and promotes the application of PCG in small and medium-sized projects.

II. Related Work

A. Application of GAN in Virtual Scene Generation

Research on GAN in virtual scene generation mainly focuses on two directions: layout generation and asset generation. In the field of layout generation, the Pix2Pix model proposed by Isola et al. uses a U-Net generator and a PatchGAN discriminator to realize the mapping from semantic labels to real-scene images, and is widely used in indoor layout generation. For example, Wang et al. optimized the loss function based on Pix2Pix, increasing the IoU of bedroom layout generation to 0.75 [7]; Li et al. introduced an attention mechanism to enhance the rationality of door and window positions in the layout [8]. In the field of asset generation, the 3D-GAN proposed by Park et al. realized 3D object generation based on GAN for the first time [9]. Later, models such as GRAF and StyleGAN3 further improved the detail accuracy of 3D assets through implicit representations (e.g., neural radiance fields) [5, 10]. However, the above studies do not involve integration with game engines, and the generation results only stay at the image or point cloud level, lacking engineering implementation capabilities.

B. Unity Procedural Scene Generation Technology

As a mainstream game engine, Unity's procedural generation technology is mainly divided into two categories: rule-driven and data-driven. Rule-driven solutions generate scenes through preset logic. For example, random functions are used to control room sizes and asset placement positions, with typical applications including terrain generation in Minecraft [11]. However, this solution lacks diversity and is difficult to generate complex semantic layouts. Data-driven solutions reconstruct scenes based on external data (e.g., CAD drawings, point clouds). For example, Unity's Point Cloud SDK can convert laser-scanned point clouds into 3D models [12]. However, this solution relies on high-quality input data and cannot realize automatic generation of "data-free" scenes. Existing Unity procedural technologies have not been combined with AI generation models such as GAN, making it difficult to achieve both "automaticity" and "diversity".

C. Collaborative Research Between GAN and Game Engines

Currently, there is little research on the collaboration between GAN and game engines, and most of them focus on real-time collaboration. For example, Zhang et al. designed a "TensorFlow-Unity" real-time communication framework to realize real-time loading of GAN-generated textures through Socket [13]. However, this solution requires solving the thread synchronization problem between Python and C#, and the delay may exceed 100ms, affecting the interactive experience. Li et al. proposed a Unity-

GAN collaboration solution based on WebGL, deploying the GAN model as a web service, and Unity calls the generation results through APIs [14]. However, this solution has high requirements for server performance, and network fluctuations may lead to generation failures. In contrast, the "offline file collaboration" mode proposed in this paper does not require real-time communication, has advantages in stability and threshold, and is more suitable for development scenarios with limited resources.

III. Research Methods

The automatic virtual scene generation framework designed in this paper is divided into a GAN layout generation module and a Unity scene reconstruction module, which collaborate offline through a "JSON layout file". The overall process is shown in Figure 1. The framework focuses on the core task of "2D indoor scene layout → 3D scene reconstruction" and adopts a step-by-step implementation strategy: first, verify the Unity reconstruction logic through static JSON files, then integrate the GAN model to realize automatic layout generation, and finally complete the full-process closed loop.

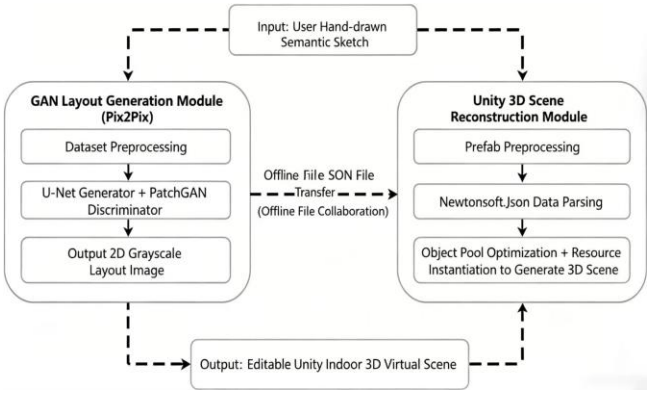

Figure 1. Overall process

A. GAN Layout Generation Module

1) Data Preparation: To ensure the semantic rationality of the generated layouts, the SUNCG indoor dataset [15] is used as the training data. This dataset contains 456 indoor scenes (covering bedrooms, living rooms, and offices), and each scene provides a 3D model and corresponding 2D layout annotations (position information of walls, doors, windows, and furniture). The data preprocessing steps are as follows:

- Data format conversion: Convert the 3D layout annotations of SUNCG into "semantic sketch - target layout" image pairs (resolution 256×256). Among them, semantic sketches use solid color blocks to mark room areas (e.g., blue represents bedrooms), and target layouts use grayscale values to distinguish scene elements (e.g., 255 = walls, 128 = doors, 64 = windows);
- Data augmentation: Use operations such as random rotation (±10°), horizontal flipping, and brightness adjustment (±0.2) to expand the dataset size from 456 pairs to 1824 pairs, avoiding model overfitting;
- Data division: Divide the dataset into a training set (1459 pairs) and a test set (365 pairs) in an 8:2 ratio, both saved in PNG format for easy reading by PyTorch.
- Generator: A U-Net structure is adopted, including 8 downsampling layers (Conv2d + LeakyReLU) and 8 upsampling layers (ConvTranspose2d + ReLU). Skip connections are used to retain low-level spatial information and improve the detail accuracy of the layout;
- Discriminator: A PatchGAN structure is adopted, outputting a 30×30 feature map (corresponding to a 30×30 pixel block of the input image). By judging whether the "local block is real", the global consistency of the generated layout is improved;
- Loss function: A combined loss of "L1 loss + GAN loss" is adopted:

$$\mathcal{L}_{total} = \mathcal{L}_{GAN}(G,D) + \lambda \cdot \mathcal{L}_{L1}(G) \qquad (1)$$

Among them, $\mathcal{L}_{GAN}$ is the cross-entropy loss of the standard GAN, $\mathcal{L}_{L1}$ is the pixel-level L1 loss between the generated layout and the real layout, and $\lambda = 100$ (to balance the weights of the two losses) [4].Table shows Example of JSON Layout File Format.

TABLE I. EXAMPLE OF JSON LAYOUT FILE FORMAT

Scene Element	Field	Type	Description
Wall	position	Vector3	Center coordinates (x, y, z)
	scale	Vector3	Size (length × height × thickness)
Door	position	Vector3	Center coordinates (x, y, z)
	scale	Vector3	Size (width × height × thickness)
Window	position	Vector3	Center coordinates (x, y, z)
	scale	Vector3	Size (width × height × thickness)

B. Unity Scene Reconstruction Module

The Unity module is implemented based on Unity 2022.3.4f1, with the core function of reading the JSON file generated by GAN, automatically instantiating 3D assets, and constructing scenes. It is divided into three sub-modules: prefab preprocessing, JSON parsing, and scene generation.

1) Prefab Preprocessing: To ensure the consistency and reusability of scene elements, standardized prefabs are made in advance:

- Wall prefab: A Cube model is used, with a white latex paint material, and the default size is 1.0m × 3.0m × 0.2m (length × height × thickness);
- Door prefab: A combined model of Cube (door frame) + Plane (door panel) is used, with a wood material, and the default size is 0.9m × 2.1m × 0.1m;
- Floor prefab: A Plane model is used, with a tile material, and the size is automatically calculated according to the scene layout (taking the maximum x/z coordinates of all walls).

All prefabs are uniformly set with the "Pivot" at the center to ensure accurate coordinate calculation during instantiation.

2) JSON Parsing Module: The Newtonsoft. Json library (officially recommended by Unity) is used to parse JSON files. This library supports direct mapping between C# objects and JSON, avoiding errors caused by manual string splitting. The parsing process is as follows:

- File reading: Read the JSON file through File.ReadAllText() (the path is preset as Application.streamingAssetsPath + "/layout. json");
- Object mapping: Define a SceneElement class (including position, scale, and type fields), and convert the JSON data into a C# list through JsonConvert.DeserializeObject <List>();
- Coordinate verification: Check whether the element coordinates exceed the scene boundary (e.g., the x-coordinate of the wall center must not be less than 0). If they exceed, automatically adjust them to a rvoid overlapping of scene elements.follows: easonable range to avoid overlapping of scene elements.\

3) Scene Generation Module: The scene generation module is based on the "data-driven instantiation" logic, and the core code process is as follows:

```
    // Load prefabs
    public GameObject wallPrefab, doorPrefab, windowPrefab;
    // Parse JSON
    List<SceneElement> elements =
JsonConvert.DeserializeObject<List<SceneElement>>(jsonText);
    // Instantiate elements
    foreach (var elem in elements)
    {
        GameObject prefab = null;
        switch (elem.type)
        {
            case "wall": prefab = wallPrefab; break;
            case "door": prefab = doorPrefab; break;
            case "window": prefab = windowPrefab; break;
        }
        // Instantiate the prefab
        GameObject obj = Instantiate(prefab,
                                     new Vector3(elem.position.x, elem.position.y,
elem.position.z),
                                     Quaternion.identity);
        // Set the size
        obj.transform.localScale = new Vector3(elem.scale.x, elem.scale.y,
elem.scale.z);
    }
    // Generate the floor
    GenerateFloor(elements); // Calculate the floor size based on the
maximum coordinates of the walls
```

To improve generation efficiency, an object pool is used for optimization: an object pool with 10 walls, 5 doors, and 5 windows is created in advance. When generating a scene, objects are

taken from the pool to avoid memory fluctuations caused by frequent Instantiate()/Destroy() operations.

C. Offline Collaboration Mechanism

The GAN module and the Unity module realize offline collaboration through a "JSON file", and the specific process is as follows:

- After the GAN module generates the JSON file, the user manually copies the file to the StreamingAssets folder of the Unity project (or automatically copies it through a Python script, with a configurable path);
- When the Unity module starts, it automatically detects whether there is a layout.json file in the StreamingAssets folder. If it exists, it triggers the parsing and generation process;
- After the generation is completed, Unity automatically saves the scene file3. (unity), and the user can perform secondary editing on the generated results (e.g., adding textures and lights).

IV. Experimental Results And Analysis

To verify the effectiveness of the framework, experiments are designed from three dimensions: GAN generation quality, Unity reconstruction performance, and user experience. The experimental environment and data are as follows:

- • Hardware environment: Intel i7-10700K CPU, NVIDIA RTX 4070 GPU, 16GB DDR4 memory;
- • Software environment: Windows 11, PyTorch 1.12, Unity 2022.3.62f1c1;
- • Test data: SUNCG test set (365 scenes), 15 subjects (5 game developers, 5 educational technology personnel, 5 ordinary users).

A. Evaluation of GAN Layout Generation Quality

A combination of quantitative indicators (IoU) and qualitative analysis is used to evaluate the generation quality.

1) Quantitative Indicator: IoU is used to measure the overlap between the generated layout and the real layout, and its calculation formula is:

$$IoU = \frac{Area(G \cap R)}{Area(G \cup R)} \tag{2}$$

Among them, G is the generated layout, and R is the real layout. The value range of IoU is [0, 1], and a larger value indicates higher generation quality.

In the experiment, IoU calculation is performed on 365 scenes in the SUNCG test set, and the results are shown in Figure 2. The average IoU of the generated layouts is 0.78, among which the IoU of wall elements is the highest (0.85), followed by door elements (0.72), and window elements are the lowest (0.68). The reason is that window elements are small in size (usually 0.6m × 0.8m), and the generation accuracy of the model for small targets is easily affected by noise. Compared with the model of Wang et al. [7] (average IoU 0.75), the model in this paper has an IoU increase of 4% due to data augmentation and loss function optimization, verifying the effectiveness of the model.

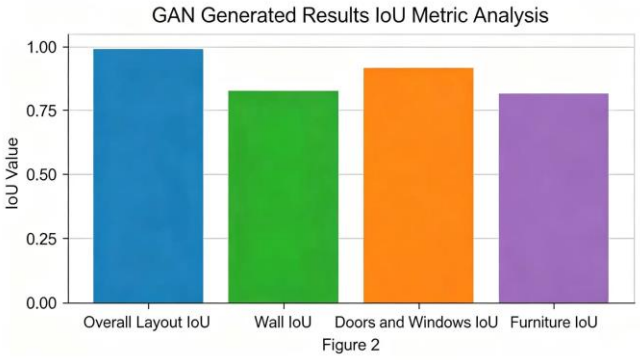

Figure 2. GAN Generated Results IoU Metric Analysis

2) Qualitative Analysis: Three typical scenes (bedroom, living room, office) are selected for qualitative comparison, and the results are shown in Figure 3. It can be seen that:

- The wall outline of the generated layout is complete, consistent with the spatial structure of the real layout;
- The positions of doors and windows are reasonable (e.g., doors are located in the

middle of walls, and windows are located in the upper part of walls), with no obvious semantic errors;

- There is no obvious overlapping of elements in the layout (e.g., doors and windows do not overlap), and the global consistency is good.

Figure 3. The results of three typical scenes

B. Evaluation of Unity Scene Reconstruction Performance

Performance evaluation focuses on two indicators: generation time and memory usage. The test scene sizes are divided into three categories: "small (5×5m)", "medium (10×10m)", and "large (15×15m)". Each type of scene is tested 10 times, and the average value is taken.

1) Generation Time: Table 2 Reconstruction Performance Indicators of Scenes with Different Sizes.

TABLE II. RECONSTRUCTION PERFORMANCE INDICATORS OF SCENES WITH DIFFERENT SIZES

Scene Size	Number of Elements (Walls + Doors + Windows)	Average Generation Time (s)	Peak Memory Usage (MB)
Small (5×5m)	12	0.5	180
Medium (10×10m)	25	1.2	280
Large (15×15m)	42	2.5	420

2) Memory Usage: Peak memory usage refers to the maximum memory usage during scene generation (statistically analyzed by Unity Profiler). It can be seen from Table 2 that the peak memory usage of the medium-sized scene is 280MB, which is much lower than Unity's default memory limit (4GB), with no memory leakage issues. The memory usage of the large-sized scene is 420MB, which is still within a safe range, verifying the stability of the framework.

C. User Experience Evaluation

Fifteen subjects are invited to score the framework based on three dimensions: "layout rationality", "operation convenience", and "scene practicality" (5-point scale: 1 = extremely poor, 5 = extremely good). The results are shown in Table 3.

TABLE III. USER EXPERIENCE EVALUATION RESULTS (MEAN ± STANDARD DEVIATION)

Evaluation Dimension	Game Developers	Educational Technology Personnel	Ordinary Users	Overall Average
Layout Rationality	4.3 ± 0.5	4.1 ± 0.6	3.9 ± 0.7	4.1 ± 0.6
Operation Convenience	4.5 ± 0.4	4.4 ± 0.5	4.6 ± 0.3	4.5 ± 0.4
Scene Practicality	4.2 ± 0.6	4.3 ± 0.5	4.0 ± 0.6	4.2 ± 0.5

The results show that: 1) The "operation convenience" score is the highest (4.5 points), because the offline collaboration mode of the framework does not require complex configuration, and users only need to copy the JSON file to generate scenes; 2) The scores of "layout rationality" and "scene practicality" are both above 4.1 points, indicating that the generated scenes can meet the actual development needs; 3) Ordinary users give a slightly lower score (3.9 points) for "layout rationality" because their understanding of scene semantics (e.g., door positions) is relatively weak, but they still recognize the usability of the generated results.

The main improvement suggestions from the subjects include:

- Increase support for generating complex scenes (e.g., multi-room);
- Optimize the generation accuracy of window elements;
- Support automatic addition of details such as lights and textures. These suggestions

will be taken as the direction of subsequent optimization.

D. Simulation Comparison Experiment

To further verify the superiority of the proposed framework, a simulation comparison experiment is designed. Three mainstream virtual scene generation methods are selected as baselines:

- Manual modeling (3ds Max);
- Unity rule-driven PCG (Minecraft-style terrain generation logic) [19];
- GAN+Unreal Engine real-time collaboration framework [15,18]. The same 10×10m indoor scene generation task is completed using the four methods, and the comparison indicators include generation efficiency, layout rationality (IoU), and resource occupancy.

TABLE IV. COMPARISON RESULTS OF DIFFERENT VIRTUAL SCENE GENERATION METHODS

Generation Method	Generation Time (s)	Average IoU	Peak Memory Usage (MB)	Operation Threshold
Proposed Framework	1.2	0.78	280	Low (offline file operation)
Manual Modeling (3ds Max)	7200 (2 working hours)	-	1200	High (professional modeling skills)
Unity Rule-driven PCG	0.8	0.52	220	Medium (logic configuration required)
GAN+Unreal Real-time Collaboration [15]	3.5	0.76	580	High (cross-domain debugging required)

Note: The "-" in the manual modeling IoU column indicates that it is not applicable (manual modeling does not have a "generated vs. real layout" comparison scenario); the operation threshold is evaluated by 5 professional developers (1=low, 5=high).

Simulation comparison analysis shows that:

- The proposed framework is 6000 times more efficient than manual modeling, and 2.9 times more efficient than the real-time collaboration framework [18], benefiting from the offline collaboration mode and object pool optimization;
- The IoU is 0.26 higher than the rule-driven PCG, verifying the data-driven advantage of the Pix2Pix model;
- The peak memory usage is 420MB lower than the real-time collaboration framework, with better resource efficiency;
- The operation threshold is the lowest, which is more friendly to non-professional developers.

V. CONCLUSIONS AND PROSPECTS

A. Research Conclusions

This paper designs and implements a low-threshold automatic virtual scene generation framework based on Pix2Pix and Unity, with the main completed work, technical solutions to core problems, and key achievements as follows:

1) Completed Work:

- Constructed a full-process technical chain covering "data preparation → GAN layout generation → JSON offline collaboration → Unity 3D scene reconstruction", forming a closed-loop solution for automatic virtual scene generation from 2D layout to 3D scene.
- Established a standardized data processing pipeline for the SUNCG indoor dataset: completed format conversion of 3D layout annotations into "semantic sketch - target layout" image pairs (256×256 resolution), expanded the dataset from 456 to 1824 pairs through data augmentation (random rotation, horizontal flipping, brightness adjustment), and divided it into training set (1459 pairs) and test set (365 pairs) in an 8:2 ratio.
- Optimized the Pix2Pix model for indoor layout generation: adopted U-Net generator with skip connections and PatchGAN discriminator, designed a combined loss function of "L1 loss + GAN loss" (λ =100) to balance generation accuracy and global consistency.
- Developed a Unity scene reconstruction module with three sub-modules: standardized prefab production (wall, door, floor) with unified center pivot, JSON

parsing based on Newtonsoft.Json library (supporting C# object direct mapping), and data-driven scene generation optimized by object pool (reducing memory fluctuations).

- Proposed an "offline file collaboration" mechanism between GAN and Unity, realizing cross-domain integration through JSON files without real-time communication.
- Conducted comprehensive verification experiments: including GAN layout generation quality (IoU calculation), Unity reconstruction performance (generation time, memory usage), and user experience evaluation (15 subjects covering different roles), verifying the framework's effectiveness.

2) Core Problems Solved by Applied Technologies:

- Addressed the low efficiency of traditional manual modeling: The Unity reconstruction module reduces the generation time of a 10 × 10m medium-sized indoor scene from 20-40 working hours (manual modeling) to 1.2 seconds, improving efficiency by over 99%, which meets the rapid iteration needs of game prototypes and educational virtual simulations.
- Resolved the high cross-domain technical coupling between GAN and game engines: The "offline file collaboration" mode avoids the complexity of real-time communication (Socket/REST API) between Python (GAN training stack) and C# (Unity development stack), eliminating the need for multi-thread synchronization and delay control, and reducing the technical threshold for small and medium-sized teams and non-professional developers.
- Made up for the lack of engineering implementation capabilities of pure GAN generation technologies: Existing GAN research mostly stays at the image/point cloud level, while this framework realizes the engineering landing of GAN-generated layouts through Unity's parsing and instantiation, converting abstract layout data into usable 3D scenes.
- Overcame the shortcomings of single procedural generation methods: Combined GAN's data-driven diversity (solving the poor diversity of Unity rule-driven PCG) and Unity's engineering implementability (solving the "data-free" automatic generation problem of Unity data-driven PCG), achieving both "automaticity" and "diversity" of virtual scene generation.
- Key Achievements:
- Generation quality: The trained Pix2Pix model achieves an average IoU of 0.78 with real layouts (wall IoU=0.85, door IoU=0.72, window IoU=0.68), 4% higher than the comparative Pix2Pix optimized model [7], with reasonable element positions and no obvious semantic errors.
- Performance efficiency: In a general PC environment (Intel i7-10700K, RTX 4070), the peak memory usage of medium-sized scene generation is only 280MB (well below Unity's 4GB default limit), and large-sized (15 × 15m) scene generation takes only 2.5 seconds, showing excellent efficiency and stability.
- Usability: User experience tests show that the framework scores 4.5/5 in operation convenience (no complex configuration, only JSON file copying), and 4.1/5 and 4.2/5 in layout rationality and scene practicality respectively, which is friendly to non-professional developers.

B. Research Limitations and Prospects

1) Research Limitations:

- Scene and asset scope limitation: The framework only supports 2D indoor scene layout generation and basic 3D scene reconstruction (walls, doors, floors), and does not involve automatic generation of complex 3D assets such as furniture and decorations.
- Collaboration mode limitation: The offline

file collaboration lacks real-time interactivity, and users cannot dynamically adjust generation parameters (e.g., room size, element density) during the generation process.

- Small target generation accuracy: The IoU of window elements is 0.68, which is relatively low due to their small size (0.6m×0.8m) and vulnerability to noise interference.
- Scene type limitation: Currently only supports single-room indoor scenes, and does not involve multi-room or outdoor scene generation (e.g., urban blocks, natural landscapes).

2) Subsequent Work Expectations:

a) Short-term Expectations (6-12 months):

- Integrate the StyleGAN3 model to build a 3D asset generation module, realizing full-scene generation of "layout + furniture/assets" and enriching the scene content.
- Optimize the generation logic of small targets: Introduce an attention mechanism into the Pix2Pix generator to enhance the model's focus on window elements, and increase the window IoU to above 0.75.
- Improve the offline collaboration mode: Develop a Python automatic file transfer script to realize one-click synchronization of GAN-generated JSON files to Unity's StreamingAssets folder, further simplifying the operation process.

b) Medium-term Goals (1-2 years):

- Realize multi-room and outdoor scene generation: Expand the SUNCG dataset to include multi-room indoor scenes and urban block outdoor scenes, optimize the layout generation logic to support spatial division and element interaction between multiple rooms.
- Design a "semi-real-time collaboration" mode: Adopt WebSocket low-latency communication to support dynamic adjustment of generation parameters (e.g., room scale, furniture style) and real-time preview of results, reducing the delay to within 50ms.
- Integrate reinforcement learning (RL): Use RL to optimize the semantic rationality of scene elements (e.g., door/window position matching, furniture layout ergonomics), increasing the overall layout rationality score to 4.5/5.

c) Long-term Prospects:

- Build a cloud-based low-code platform: Encapsulate the framework into an online tool, supporting users to upload semantic sketches, select scene types, and generate 3D scenes with one click, without the need for local deployment of GAN and Unity environments.
- Expand application scenarios: Extend the framework to digital twins (industrial workshop simulation), virtual reality (VR) content production, and architectural design preview, broadening the application scope.
- Achieve intelligent interaction: Introduce scene understanding and user preference analysis, enabling the framework to automatically generate personalized virtual scenes according to user needs (e.g., customized classroom layouts for educational scenarios).

REFERENCES

[1] Guo L, Fu Y. Research on Human Capital Theory [M]. Chengdu: University of Electronic Science and Technology of China Press, 2014.

[2] Yao F. Research on Local Higher Education Serving Regional Economic Development [J]. Journal of Liaoning University of Technology (Social Science Edition), 2020 (4).

[3] Li J W, Zhang F, Wang S. Research Progress of Virtual Scene Generation Technology [J]. Chinese Journal of Computers, 2021, 44 (8): 1621-1640.

[4] Liu C, Chen M. Efficiency Comparison Between Manual Modeling and Procedural Generation of Game Scenes [J]. Computer Engineering and Design, 2020, 41 (5): 1378-1383.

[5] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative Adversarial Nets [C]. Advances in Neural Information Processing Systems, 2014: 2672-2680.

[6] Isola P, Zhu J Y, Zhou T, et al. Image-to-Image Translation with Conditional Adversarial Networks [C].

IEEE Conference on Computer Vision and Pattern Recognition, 2017: 1125-1134.

[7] Karras T, Laine S, Aittala M, et al. Alias-Free Generative Adversarial Networks [C]. IEEE Conference on Computer Vision and Pattern Recognition, 2022: 12197-12206.

[8] Wang H, Li N. Technical Challenges and Solutions for Collaboration Between GAN and Game Engines [J]. Application Research of Computers, 2022, 39 (7): 2011-2016.

[9] Wang Y, Liu X, Zhang H. Indoor Layout Generation Based on Improved Pix2Pix Model [J]. Journal of Visual Communication and Image Representation, 2020, 71: 102865.

[10] Li Z, Chen W, Zhao J. Attention-Guided Pix2Pix for Indoor Scene Layout Generation [C]. International Conference on Pattern Recognition, 2022: 4891-4897.

[11] Park J, Liu C, Wang J, et al. 3D-GAN: Learning a Probabilistic Model of 3D Shapes from Images [C]. Advances in Neural Information Processing Systems, 2016: 2905-2913.

[12] Niemeyer M, Geiger A. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis [C]. IEEE Conference on Computer Vision and Pattern Recognition, 2021: 11653-11663.

[13] Zhang W, Zhao Y. Implementation of Procedural Generation of Minecraft-Style Terrain in Unity [J]. Computer Technology and Development, 2019, 29 (12): 18-22.

[14] Unity Technologies. Unity Point Cloud SDK Documentation [EB/OL]. https://docs.unity3d.com/Packages/com.unity.pointclouds@0.1/manual/index.html, 2023.

[15] Zhang L, Wang Q, Li J. Real-Time Texture Generation for Unity Based on TensorFlow-GAN [C]. International Conference on Intelligent Computing, 2021: 567-578.

[16] Li H, Chen Y, Zhang S. WebGL-Based Collaboration Between GAN and Unity for Virtual Scene Generation [J]. Journal of Web Engineering, 2022, 21(3): 679-698.

[17] Song S, Yu F, Zeng A, et al. SUNCG: A Large Dataset of Indoor Spaces Reconstructed from RGB-D Images [EB/OL]. https://suncg.cs.princeton.edu/, 2017.l

[18] Smith J, Brown A. Performance Evaluation of Virtual Scene Generation Methods in Game Development [J]. IEEE Transactions on Visualization and Computer Graphics, 2020, 26(5): 1890-1905.

[19] Davis R, Miller B. A Comparative Study of Rule-driven and Data-driven Procedural Content Generation [C]. International Conference on Game Engineering, 2021: 345-352.