City$\mathcal{X}$: Controllable Procedural Content Generation for Unbounded 3D Cities

Shougao Zhang[2*] Mengqi Zhou[1*] Yuxi Wang[1,3] Chuanchen Luo[1,3]
Rongyu Wang[1] Yiwei Li[1] Xucheng Yin[4] Zhaoxiang Zhang[1,3†]
Junran Peng[3,4‡]

[1]University of Chinese Academy of Sciences, [2]China University of Geosciences (Beijing)
[3]Center for Research on Intelligent Perception and Computing, CASIA, [4]University of Science and Technology Beijing
zhangshougao@email.cugb.edu.cn, jrpeng4ever@126.com, mengqi.zhou@sensetime.com
yuxi.wang@ia.ac.cn, zhaoxiang.zhang@ia.ac.cn

Abstract

Generating a realistic, large-scale 3D virtual city remains a complex challenge due to the involvement of numerous 3D assets, various city styles, and strict layout constraints. Existing approaches provide promising attempts at procedural content generation to create large-scale scenes using Blender agents. However, they face crucial issues such as difficulties in scaling up generation capability and achieving fine-grained control at the semantic layout level. To address these problems, we propose a novel multi-modal controllable procedural content generation method, named City$\mathcal{X}$ which enhances realistic, unbounded 3D city generation guided by multiple layout conditions, including OSM, semantic maps, and satellite images. Specifically, the proposed method contains a general protocol for integrating various PCG plugins and a multi-agent framework for transforming instructions into executable Blender actions. Through this effective framework, City$\mathcal{X}$hows the potential to build an innovative ecosystem for 3D scene generation by bridging the gap between the quality of generated assets and industrial requirements. Extensive experiments have demonstrated the effectiveness of our method in creating high-quality, diverse, and unbounded cities guided by multi-modal conditions. Our project page: `https://cityx-lab.github.io/`.

1 Introduction

Recently, 3D generative models have witnessed remarkable achievements in object generation, scene creation, and human avatars, which are crucial technologies for game development, virtual reality, animation, and film production. For example, DreamFusion [1] and Magic3D [2] produce 3D objects guided based on text instructions, while Zero123 [3], Wonder3D [4] and SV3D [5] transform 2D images into high-quality 3D assets. Different from these object-centric 3D generation works, we focus on a challenging unbounded city-scale scene generation, which involves numerous 3D assets (*buildings, roads, vegetation, rivers, etc.*), various city styles (*modern style, traditional style*), and strict layout constraint. Although existing pioneers [6, 7, 8, 9] have attempted to generate larger city-scale scenes, a significant gap remains between the quality of the generated content and the standards required for industrial applications.

[*]Equal contributions.
[†]Corresponding author.
[‡]Corresponding author.

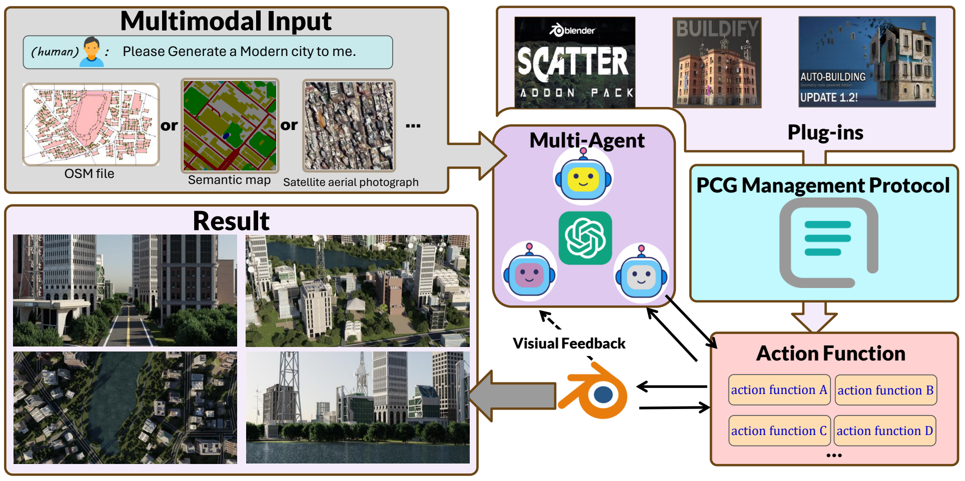

Figure 1: The proposed City$\mathcal{X}$, under the guidance of multimodal inputs including OSM data, semantic maps, and satellite images, facilitates the automatic creation of realistic large-scale 3D urban scenes.

Typically, previous approaches attempt to generate realistic large-scale 3D virtual cities from three aspects: generative methods [6, 7, 8], NeRF-based city synthetic [10, 11, 12], and procedural content generation methods [9]. Specifically, CityGen [7], InfiniCity [6], and CityDreamer [8] focus on lifting 2D knowledge to 3D city at layout-level, building-level and block-level. MegaNeRF [11] and BlockNerf [12] achieve realistic city-scale scene synthesis by expanding NeRF [13] and its variants to render city scenes. However, due to the complexity of city generation and the scarcity of training data, the challenges of city-scale 3D scene creation have not been thoroughly investigated.

Notably, existing works [14, 15, 9] have explored the use of Blender agents driving procedural content generation (PCG) to create large-scale scenes. These methods usually leverage the knowledge of pre-trained large language models (LLMs) to produce executable codes. Benefiting from the powerful task-planning capabilities of LLMs and the rule-based generation process of PCG, these approaches provide a promising way to bridge the gap between generated assets and industrial requirements. However, due to the irregular, non-uniform, and extensive nature of most accessible Blender plugins, previous methods [9, 14] face challenges in scaling up and ensuring the generalization. For instance, CityGen3D [4] and SceneCity [5] are two commonly used city generation plugins in Blender, which have their own specific blueprints rules and node parameters. Previous works usually require the design of customized agents for both CityGen and SceneCity, which significantly limits their adaptability to different plugins. Moreover, as these methods only support text instruction inputs, achieving fine-grained controlled generation, especially for layout-constrained generation, remains challenging. Therefore, we argue existing Blender agents still face two crucial challenges, beyond merely producing executable codes. *1. How to achieve seamless adaptation to various existing plugins at no additional cost,* which is essential for the entire community and for scaling up generation capabilities. *2. How to enable precise control in urban scene generation,* such as detailed adjustments based on semantic maps or OpenStreetMap (OSM) data.

In this paper, we introduce a multi-modal controlled method, named City$\mathcal{X}$, to achieve high-quality 3D city generation using Blender-executable Python scripts, as shown in Fig. 1. To tackle integration challenges among various plugins, we first provide a general protocol to reformat the interface of different assets. Similar to excellent previous work HuggingGPT [16], the proposed protocol shows the potential to build an ecosystem for the community, which is crucial for adapting all kinds of PCG plugins following our rule. Specifically, the protocol primarily comprises three parts: (i) a dynamic API conversion interface, enabling effective and flexible integration of different action functions; (ii) structured encapsulation, accelerating the encapsulation process for beginners and reducing the barriers to using PCG; (iii) infinite asset library and asset retrieval, facilitating unlimited

[4]https://citigen.gumroad.com/l/CITIGEN
[5]https://www.cgchan.com/store/scenecity?tdsourcetag=s_pctim_aiomsg

expansion of assets to meet the diverse needs of scene generation. On the other hand, we propose a multi-agent framework effectively to manage the complex, multi-round interactions between large language models (LLMs) and Blender with visual feedback. Through these components, City$\mathcal{X}$ can produce high-quality 3D urban scenes based on various input guidance, including semantic maps, XML-format OSM data, satellite images and textual descriptions.

The contributions of this paper are summarized as follows:

1. We propose a general protocol for LLM agents to integrate various PCG plugins, which shows the potential for building an innovative ecosystem of 3D scene generation.

2. We introduce a controllable 3D city generation framework named City$\mathcal{X}$, which can produce action codes for Blender by the proposed multi-agent framework with visual feedback.

3. The proposed City$\mathcal{X}$ can generate high-quality and diversity unbounded 3D cities under the guidance of multi-modal inputs, including OSM, semantic maps, and satellite images.

2 Related Works

Generating a 3D urban scene is a complex task involving multiple modules, such as accurately generating a reasonable layout and then constructing appropriate instances on the layout. Additionally, applying Multi-Agent systems to complex tasks like 3D urban scene generation presents significant challenges. In this section, we will discuss works related to these aspects.

Agent Systems Based on LLMs. When researching agent systems based on Large Language Models (LLMs), the focus lies on effectively integrating and applying these models to execute complex tasks. Existing relevant work encompasses various aspects, including task management, role-playing, dialogue patterns, and tool integration. For example, [17] utilizes the expansive domain knowledge of LLMs on the internet and their emerging zero-shot planning capabilities to execute intricate task planning and reasoning. [18] investigates the application of LLMs in scenarios involving multi-agent coordination, covering a range of diverse task objectives. [19] presents a modular framework that employs structured dialogue through prompts among multiple large pretrained models. Moreover, specialized LLMs for particular applications have been explored, such as HuggingGPT [16] for vision perception tasks, VisualChatGPT [20] for multi-modality understanding, Voyager [21] and [22], SheetCopilot [23] for office software, and Codex [24] for Python code generation. Furthermore,AutoGPT[25] demonstrates the ability to autonomously complete tasks by enhancing AI models, but it is a single-agent system and does not support multi-agent collaboration. In contrast, BabyAGI[26] uses multiple agents to manage and complete tasks, with each agent responsible for different task modules such as creating new tasks, prioritizing the task list, and completing tasks. Multi-agent debate research includes works[27][28] indicating that debates among multiple LLM instances can improve reasoning and factuality, but these methods typically lack the flexibility of tool and human involvement. AutoGen[29], as a general infrastructure, supports dynamic dialogue modes and a broader range of applications, demonstrating potential in advancing this field.

3D Urban Scene Generation. Scene-level content generation presents a challenging task, unlike the impressive 2D generative models primarily targeting single categories or common objects, due to the high diversity of scenes. Semantic image synthesis, as exemplified by [30, 31, 32, 33], has shown promising results in generating scene-level content in the wild by conditioning on pixel-wise dense correspondence, such as semantic segmentation maps or depth maps. Recent works such as [34, 35, 6] have realized infinite-scale 3D consistent scenes through unbounded layout extrapolation. Additionally, in-depth research has been conducted on using procedural content generation (PCG) techniques to generate natural scenes [36, 37] and urban scenes [38, 39, 40, 41]. For example, PMC [42] proposed a procedural method based on 2D ocean or city boundaries to generate cities. It employs mathematical algorithms to generate blocks and streets and utilizes subsequent techniques to generate the geometric shapes of buildings. While traditional computer graphics methods can generate high-quality 3D data, all parameters must be predefined during the procedural generation process. Since the generated 3D data is subject to rule limitations and exhibits a certain degree of deviation from the real world, this significantly constrains its flexibility and practical utility.

Table 1: The proportion of API Input/Output Formats and Conversion Interface Statistics. A statistical overview of API input/output formats (in percentages) is presented on the left, where PIDF represents the proportion of input data format types and PODF represents the proportion of output data format types, while the conversion interface along with its descriptions is detailed on the right.

Data Format	PIDF(%)	PODF(%)	Conversion Interface	Description
Scene Layout	0	19.05	Point_to_face_conversion	Converting points to faces.
Noise Function	0.45	0	Cube_generation	Generating cubes.
Image	0.90	0	Point_generation	Generating points.
Texture Material	0.90	32.54	Line_generation	Generating lines.
Geographic Information Data	1.35	0	Face_generation	Generating faces.
Point	3.15	0	Line_to_face_conversion	Converting lines to faces.
Boolean Value	4.50	0	Asset_placement	Placing assets within a scene or environment.
Complex Geometry	4.50	43.65	Point_to_line_conversion	Converting points to lines.
Color	5.86	0	OSM_file_retrieval	Retrieving OpenStreetMap (OSM) files.
Basic Geometry	7.66	0	Object_meshing	Meshing objects.
String Information	8.11	0	Texture_information_extraction	Extracting texture information.
Surface	12.61	2.38	Asset_material_retrieval	Retrieving material data for assets.
Line	13.96	2.38	Asset_mesh_retrieval	Retrieving mesh data for assets.
Random Number	36.04	0	Scene_object_information_extraction	Extracting scene object information.

3 Methods

Our proposed City$\mathcal{X}$ can generate highly realistic large-scale urban scenes based on multi-modal input. It consists of two key components: a PCG management protocol and a multi-agent framework. The PCG management protocol provides a universal standard to regulate various PCG plugins, enabling City$\mathcal{X}$ to easily scale up by integrating all kinds of irregular and non-uniform plugins. At the same time, the proposed multi-agent framework effectively manages the complex, multi-round interactions between large language models (LLMs) and Blender. Through these components, City$\mathcal{X}$ can produce high-quality 3D urban scenes based on multimodal input guidance, including OSM data, semantic maps, satellite images, and textual descriptions.

3.1 PCG Management Protocol

To facilitate flexible, efficient, and straightforward usage of PCG, we propose a universal protocol for managing PCG plugins, serving as a bridge connecting the LLM and Blender. The protocol primarily comprises three parts: (i) a dynamic API conversion interface, enabling effective and flexible integration of different action functions; (ii) structured encapsulation, accelerating the encapsulation process for beginners and reduces the barriers to using PCG; (iii) infinite asset library and asset retrieval, facilitating unlimited expansion of assets to meet the diverse needs of scene generation.

Dynamic API Conversion Interface. The lack of a unified communication protocol among PCG APIs not only hinders the free combination of different PCGs, thus reducing the diversity of generated content, but also requires manual adjustment of the PCG workflow, which is not user-friendly for 3D modeling beginners. To address this issue, we provide a Dynamic API Conversion Interface. Specifically, we first compile all the input and output formats of the APIs, as depicted in Table 1 on the left. Then, after summarizing all the formats, we define a comprehensive and self-consistent dynamic API conversion interface as depicted in Table 1 on the right. This Dynamic API Conversion Interface serves as a bridge connecting different APIs, providing communication interfaces for many different formats of APIs. By dynamically adjusting these interfaces, the goal of freely combining different PCGs can be achieved.

Structured Encapsulation. Since LLM cannot directly utilize PCG through Blender, PCG needs to be encapsulated into action functions for LLM execution. However, encapsulating PCG into action functions poses significant technical barriers for beginners, particularly in terms of both coding knowledge and 3D modeling expertise. To enable beginners to quickly and easily create their own action function, we propose a method of structured encapsulation. For each PCG, the encapsulation, denoted as Encapsulation S, follows a similar structure defined as $S_{\mathrm{i}}(C_{\mathrm{i}}, D_{\mathrm{i}}, I_{\mathrm{i}}, L_{\mathrm{i}}, R_{\mathrm{i}})$, where i distinguishes the i-th action function. The components of the protocol are defined as follows:

- **Classname** The name of the action function, used for indexing the action function.
- **Description** A detailed objective description of an action function.
- **Input** A detailed objective description of the input to the action function.
- **Limitation** An objective description of the functional constraints of the action function.

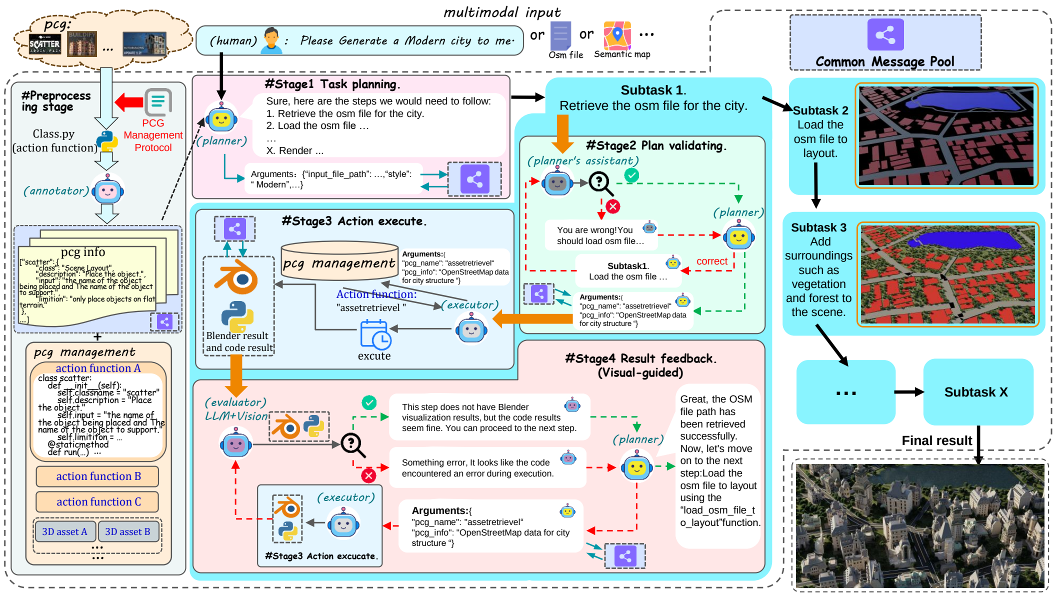
multimodal input
pcg:
(human): Please Generate a Modern city to me.
or
Osm file
or
Semantic map
Common Message Pool
#Preprocessing stage
Class.py (action function)
PCG Management Protocol
(annotator)
pcg info
pcg management
action function A
action function B
action function C
3D asset A
3D asset B
#Stage1 Task planning.
(planner)
Sure, here are the steps we would need to follow:
1. Retrieve the osm file for the city.
2. Load the osm file ...
X. Render ...
Subtask 1. Retrieve the osm file for the city.
#Stage2 Plan validating.
(planner's assistant)
You are wrong! You should load osm file...
correct
#Stage3 Action execute.
Blender result and code result
Action function: "assetretrievel"
(executor)
excute
#Stage4 Result feedback. (Visual-guided)
(evaluator) LLM+Vision
This step does not have Blender visualization results, but the code results seem fine. You can proceed to the next step.
Something error, It looks like the code encountered an error during execution.
Great, the OSM file path has been retrieved successfully. Now, let's move on to the next step: Load the osm file to layout using the "load_osm_file_to_layout" function.
Subtask 2 Load the osm file to layout.
Subtask 3 Add surroundings such as vegetation and forest to the scene.
Subtask X
Final result

Figure 2: Multi-agent Workflow: Detailed demonstration of collaboration and communication across various stages.

- **Run** Function name used to execute the action function and return the result.

We provide users with a straightforward encapsulation document. By acquainting themselves with the process of extracting API information from Blender, users can seamlessly follow the document's instructions to encapsulate their own functional actions.

Infinite Asset Libraries and Asset Retrieval. While many manually crafted assets are exquisite, in urban scene generation tasks, the compatibility between assets and layout is often more crucial than the intricacy of individual assets. To effectively obtain assets that best match the scene layout, we propose a method of infinite asset libraries. This involves continuously expanding the asset library through agent-driven PCG and providing textual descriptions and image renderings for each asset. To expedite and enhance the precision of asset retrieval, we leverage the pre-trained CLIP model for text-to-image retrieval of our assets, as illustrated in Figure X. Each rendered image of the asset is encoded into a normalized 768-dimensional vector, which is then compared with the embedding of the input description. The matching result is determined by calculating the cosine similarity. From the 10 most similar results, one is randomly selected and imported into the Blender scene.

3.2 Multi-Agent Framework

Due to the hierarchical nature of controllable elements in Blender, such as the blueprint for layout planning, the geometry node for asset scatter, and numerous parameters for asset placement, color, size, and height, this renders a naive LLM framework inadequate for handling Blender's complex and diverse action. To effectively utilize actions of different hierarchies for generating high-quality large-scale city scenes, we propose a multi-agent framework with visual feedback as depicted in Fig. 2. This framework mainly consists of four agents: annotator, planner, executor, and evaluator. The annotator labels all actions with multiple tags and stores the annotated action information in the common message pool. The planner formulates the overall task pipeline using user-provided textual information and determines the action required for different sub-tasks. The executor manages all action functions and uses the annotated action functions to process sub-tasks of procedural generation or asset manipulation in Blender. The evaluator assesses the correctness of the current sub-task by obtaining scene information from Blender, such as object location and rendered images.

Multimodal Input through Multi-Agent Framework. Unlike simple LLM frameworks, our Multi-Agent Framework can find feasible solutions through multiple rounds of interaction and visual feedback, meaning that with only a few plugins, the target task can be accomplished, avoiding the burden to reinvent the wheel. This also simplifies city generation through multimodal inputs. For example, when we import a semantic map into Blender, it is stored as a point cloud containing semantic information, but we only have face-based building generation actions. At this point, the Multi-Agent Framework will find a tool to convert vertices into faces, complete the conversion, and then use face-based building generation actions to accomplish the task.

Annotator Agent. For large-scale city generation tasks, a substantial number of action functions are required. Therefore, effectively managing these action functions is crucial. To ensure each agent can efficiently access all action of different hierarchies, we use the Annotator to label them.

The Annotator labels action functions in two steps: first, summarizing existing functions into consistent concepts guided by prompts; second, labeling each function based on these concepts. An action function may receive multiple labels. Once all action functions are processed, the labeled information is stored in the common message pool, enabling other agents to directly access it.

Planner Agent. We consider PCG-based city scene generation as an open-loop planning task with flexible steps. Specifically: (i) Termination of the city scene generation task depends on meeting the user's requirements; (ii) City scene generation tasks are a series of sequentially arranged action functions, where the order of these actions significantly impacts the final outcome. To address these challenges, we propose a dynamic planner. At the beginning of the task, the planner formulates a rough workflow as a reference. During the execution of specific sub-tasks, the planner plans the next action based on the current sub-task goals and the workflow, until the user's requirements are met.

When the planner directly receives user input, it needs to translate the user's intent into a series of referable executable actions with additional explanations, which will be stored in the common message pool. To achieve this, we prompt the planner to produce a preliminary action plan, serving as the workflow. Specifically, we utilize the labeled information L from the common message pool, the user input I, and the planner's guidance document D as inputs, allowing the planner to generate an ordered workflow W. This process is formalized as follows:

$$W \leftarrow \text{Planner}(L, I, D) \tag{1}$$

During the execution of the t-th sub-task, the planner needs to formulate actions required for the $(t+1)$-th sub-task. To ensure the accuracy and coherence of actions, the planner refers to the workflow. Specifically, based on the ordered workflow W, sub-task input I, the labeled information L from the common message pool, and the planner's guidance document D, the planner infers the next action A_{t+1}, where A_{t+1} stands for the action of the $(t+1)$-th sub-task.

$$A_{t+1} \leftarrow \text{Planner}(L, I, D, W). \tag{2}$$

Executor Agent. To achieve Interactive Workflow in Blender, we deploy the Executor within the Blender environment. All agents send action execution commands to the Executor through a local backend server. As mentioned in Section 3.1, we transform PCG plug-ins into executable action functions using structured encapsulation. This allows the Executor to initialize all actions flexibly. To enable agents to precisely control actions, each action function is structurally recorded in JSON format. For example, the scale_object action function is documented as follows:

```
scale_object_doc = {name: scale_object,description: Scale an object,
    parameters:{scale_factor:{type:tuple, description:scale factor},
        scaled_obj_name:{type:str, description:scaled object name}}}
```

During the execution of the t-th sub-task, the Executor uses the action document D and the sub-task input I to generate the action arguments $Arguments$. The action A_t is then executed in Blender based on the arguments. Here, A_{t+1} represents the action for the $(t+1)$-th sub-task, S_{t+1} represents the Blender state for the $(t+1)$-th sub-task, and S_t represents the Blender state for the t-th sub-task.

$$\text{Arguments} \leftarrow \text{Executor}(I, D),\ S_{t+1} \leftarrow \text{Blender}(S_t, A_t, Arguments) \tag{3}$$

Evaluator Agent. To address the limitations of textual feedback in urban scene generation tasks, we designed an Evaluator with visual feedback based on GPT-4V[43]. Specifically, we first render the generated scene as an image. Then, we provide both the image and the current sub-task text input to the Evaluator, guiding it with prompts to assess whether the scene's geometry and materials match the sub-task expectations. If the Evaluator determines that the rendered image is consistent with the sub-task text input in terms of geometry and materials, the evaluation ends. However, if the Evaluator identifies errors, it can pass this information to the Planner for improvements. This process is formalized as follows:

$$\text{R} \leftarrow \text{Evaluator}(\text{GPT-4V}(img, I, D)), \tag{4}$$

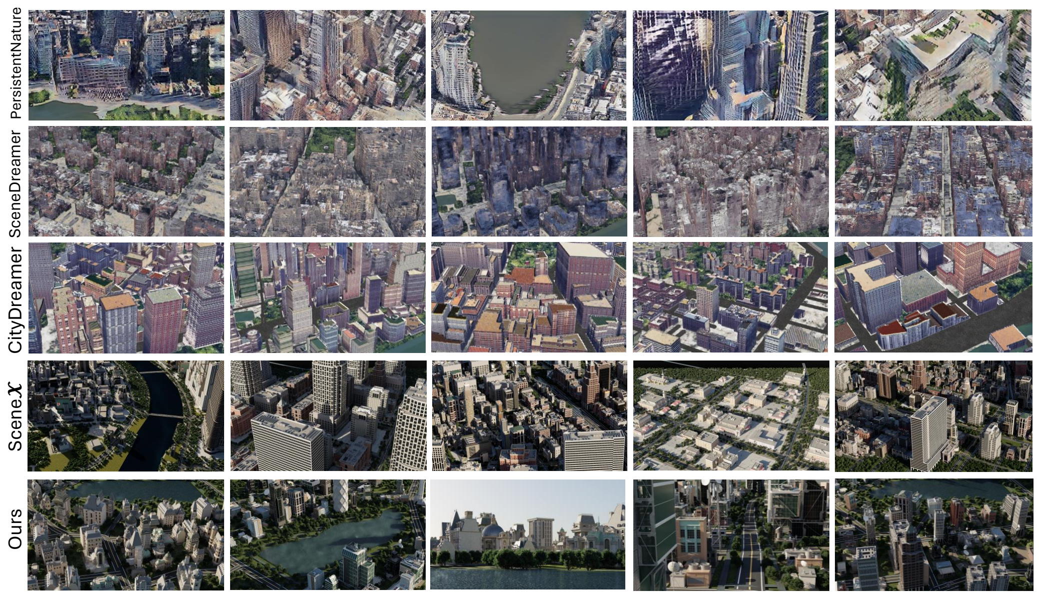

Figure 3: Comparative results on city generation. Issues with unreasonable geometry are observed in previous works, while our method performs well in generating realistic large-scale city scenes.

Table 2: Comparing the performance of different language models in city generation. The aesthetic score represents the intuitive aesthetic rating, and the rationality score is the logical coherence rating.

Model	AS		AES	
	aesthetic score	rationality score	aesthetic score	rationality score
CityDream[8]	2.80	3.05	2.65	3.40
PersistentNature[34]	1.30	1.40	1.55	1.35
SceneDreamer[35]	1.30	1.63	1.30	1.35
Scene$\mathcal{X}$[9]	3.73	3.63	3.50	3.45
City$\mathcal{X}$(Ours)	**4.30**	**4.35**	**4.15**	**4.30**

where R stands for the Evaluator result, img stands for the rendered image, I stands for the sub-task input, and D stands for the Evaluator's guidance document.

4 Experiments

The goals of our experiments are threefold: (i) to verify the capability of City$\mathcal{X}$ for generating highly realistic large-scale city with different modes of input; (ii) to prove that the Muti-Agent framework and pcg protocol we designed are effective; (iii) to compare different LLMs on the proposed benchmark.

4.1 Benchmark Protocol

Dataset. To evaluate the effectiveness of the proposed City$\mathcal{X}$, we collect 50 city Semantic Maps, 50 city Height Fields, and 50 city OSM files in XML format. We also collect 50 sets of descriptions about city styles and weather. Then, we feed them to our City$\mathcal{X}$ to generate corresponding city models, which are used to perform quantitative and qualitative comparisons.

Models. When generating and editing the 3D scenes, we adopt the leading GPT-4 as the large language model with its public API keys. To ensure stable output from the LLM, we set both the decoding temperature and the seed to 0.

Metrics: We use Executability Rate (ER@1) and Success Rate (SR@1) to evaluate the capabilities of LLMs on our City$\mathcal{X}$. The former measures the proportion of proposed actions that can be executed, and the latter is used to evaluate action correctness [44]. Additionally, we use a unified evaluation standard as a reference. We categorize the aesthetics of city scenes into five levels: Poor (1 points)/Below Average (2 points)/Average (3 points)/Good (4 points)/Excellent (5 points).

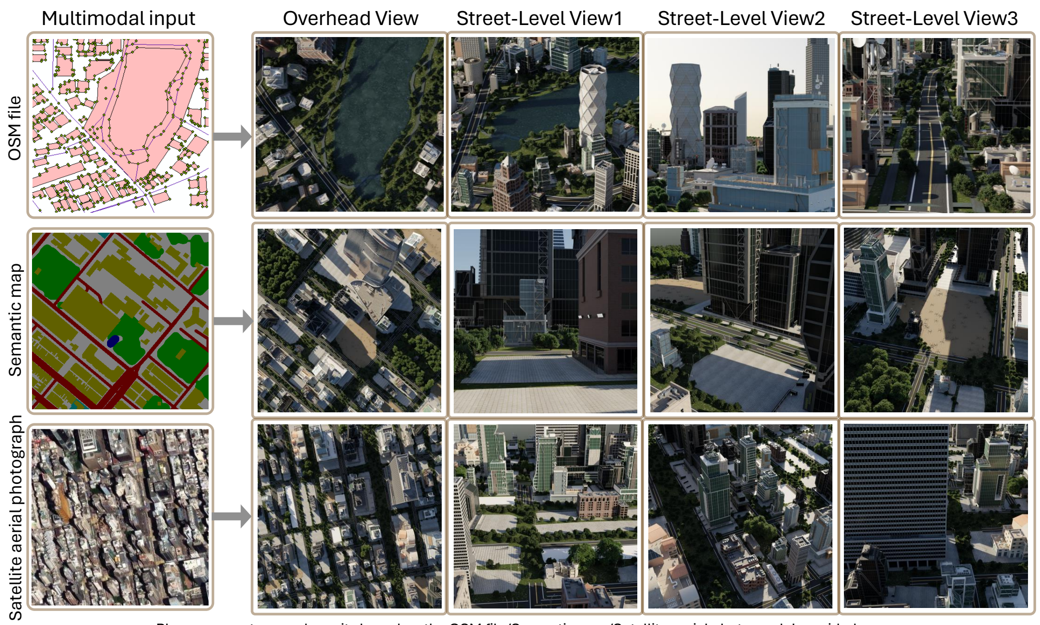
Multimodal input
Overhead View
Street-Level View1
Street-Level View2
Street-Level View3
OSM file
Semantic map
Satellite aerial photograph
Please generate a modern city based on the OSM file/Semantic map/Satellite aerial photograph I provided.

Figure 4: Urban scene generation with multimodal inputs, where we present an overhead view aligned with the multimodal input perspective, along with three street-level views.

4.2 Main Result

Urban Scene Generation with Multimodal Inputs. We first show the ability of City$\mathcal{X}$ to generate large-scale urban scenes, as depicted in Figure 4. The results show that City$\mathcal{X}$ is capable of generating highly realistic urban scenes using multimodal data inputs, including OSM data, semantic maps, and satellite images, demonstrating its effectiveness and flexibility in urban scene generation.

We also compare our method with other city generation approaches, as shown in Figure 3. The results indicate that PersistantNature[34] and InfiniCity[6] have severe deformation issues throughout the entire scene. While SceneDreamer[35] and CityDreamer[8] demonstrate improved structural consistency, their building quality remains relatively low. While Scene$\mathcal{X}$[9] achieves high quality, it encounters issues with overlapping assets and a high duplication rate of buildings. In contrast, the city generated by City$\mathcal{X}$ demonstrates a regular geometric structure and high quality, which is devoid of overlapping buildings and exhibits minimal repetition.

Aesthetic Evaluation. To better assess the quality of cities generated by City$\mathcal{X}$, we collect results from various related works on urban generation and invite 30 volunteers and 5 experts in 3D modeling to evaluate these works aesthetically. To ensure fairness, we anonymize all results. As shown in Table 2, the cities generated by City$\mathcal{X}$ attain a "Good" level in aesthetic scoring, a distinction not achieved by other works, demonstrating its highly realistic capabilities in city generation.

Specific Refinement Editing. City$\mathcal{X}$ supports specific refinement editing for scene customization, involving asset manipulation, weather adjustment, and style modification. We conduct relevant experiments, as depicted in Figure 5. Based on the results, it's clear that City$\mathcal{X}$ performs well in accurately controlling urban scenes to meet input requirements consistently.

Table 3: Ablation Study Results for Different Components of the Protocol.

Description.	Input	Limitation	ER@1	SR@1
		✓	36.00	41.67
	✓	✓	37.00	51.35
✓			44.00	56.82
✓	✓		69.00	60.87
✓		✓	73.00	61.64
✓	✓	✓	**94.00**	**82.98**

Table 4: Comparing the performance of different language models in city generation.

Model	ER@1	SR@1
Llama2-7B[45]	27.00	59.26
Mistral[46]	78.00	61.54
Gemma-2B[47]	9.00	33.33
Gemma-7B[47]	39.00	69.23
GPT-3.5-turbo[48]	72.00	75.00
GPT-4[49]	**91.00**	**81.32**

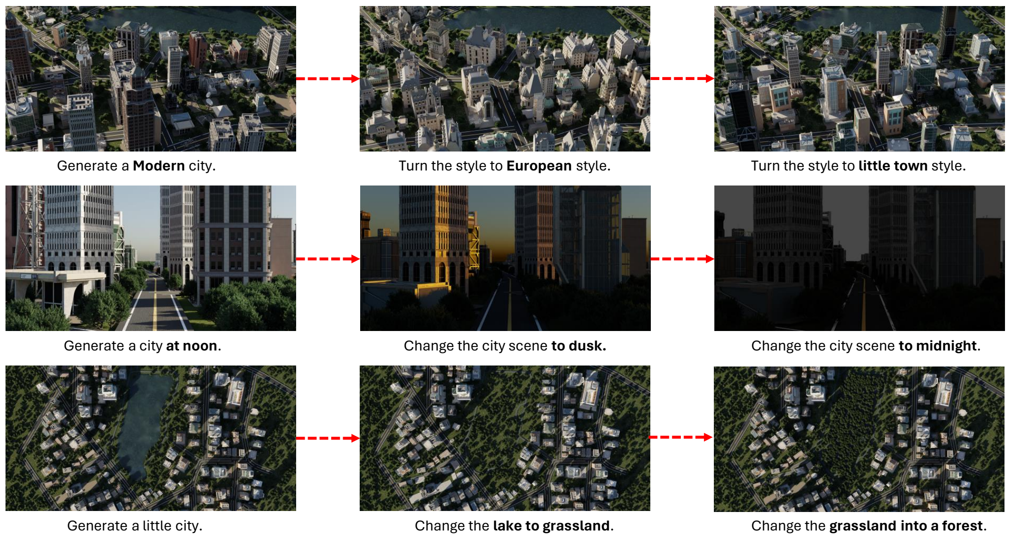
Generate a **Modern** city.
Turn the style to **European** style.
Turn the style to **little town** style.
Generate a city **at noon**.
Change the city scene **to dusk.**
Change the city scene **to midnight**.
Generate a little city.
Change the **lake to grassland**.
Change the **grassland into a forest**.

Figure 5: Performance of City$\mathcal{X}$ in scene customization and refinement editing

4.3 Ablation Study of PCG Protocol.

Analysis of Different Components. To assess the impact of each component of the structured encapsulation on the overall system, we conduct ablation experiments on individual parts of the structured encapsulation, as shown in Table 3. The table demonstrates that when encapsulating PCG, adding Description, Input, and Limitation all boost ER@1 and SR@1. Notably, including Description leads to the highest increase, with SR@1 rising by 57.00% and ER@1 by 31.63%, significantly enhancing the system's Executability Rate and Success Rate. After adding Input, SR@1 and ER@1 increase by up to 21.34% and 25.00%, respectively. Similarly, with the inclusion of Limitation, SR@1 and ER@1 see maximum increases of 22.11% and 25.00%. This suggests that incorporating Input and Limitation can improve the system's Executability Rate and Success Rate.

Comparing Agent Frameworks with Different LLMs To evaluate the effects of different large language model variants on multi-agent frameworks, we assessed the system's ER@1 and SR@1 using various LLM versions. As not all open-source models have visual perception capabilities, we used GPT-4-Vision-Preview uniformly for visual feedback. To maintain experimental stability, the temperature and seed for all LLMs were set to 0. The experiment utilized 50 description sets from Section 4.1 dataset. Results are shown in Table 4. GPT-4 achieves the highest scores in both ER@1 and SR@1, with scores of 91.00% and 89.01%, respectively. Mistral ranks second in ER@1, with a score of 78.00%, while GPT-3.5-turbo and Gemma-7B rank second and third in SR@1, with scores of 75.00% and 69.23%, respectively. It is evident that Mistral and Gemma-7B, two open-source large language models, perform comparably to GPT-3.5-turbo but still fall short of GPT-4's performance.

5 Conclusions

In this paper, we propose a novel multi-modal controllable procedural content generation method City$\mathcal{X}$ to generate realistic, unbounded 3D cities. The proposed method supports multi-modal guided conditions, such as OSM, semantic maps, and satellite images. The proposed method includes a general protocol for integrating various PCG plugins and a multi-agent framework for transforming instructions into executable Blender actions. Through this effective framework, City$\mathcal{X}$ shows potential for building an innovative ecosystem in 3D scene generation by bridging the gap between generated assets and industrial requirements. Extensive experiments have demonstrated the effectiveness of our method in creating high-quality, diverse, and unbounded cities guided by multi-modal conditions.

Societal Impacts. City$\mathcal{X}$ generates high-quality urban through the use of the PCG Management Protocol and Multi-Agent Framework, closing the gap between industrial application needs and the quality of generative models. This approach is significant for building a new ecosystem based on procedural content generation and benefits the entire PCG community.

Limitations. City$\mathcal{X}$ has two drawbacks for future improvement. Firstly, we lack an efficient method to accelerate parameter extraction in PCG, which requires significant manpower and resources. Secondly, scene generation based on PCG methods is constrained by the inherent rules of PCG itself, limiting the diversity of PCG generation techniques.

References

[1] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *The Eleventh International Conference on Learning Representations*, 2022.

[2] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023.

[3] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9298–9309, 2023.

[4] Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, et al. Wonder3d: Single image to 3d using cross-domain diffusion. *arXiv preprint arXiv:2310.15008*, 2023.

[5] Vikram Voleti, Chun-Han Yao, Mark Boss, Adam Letts, David Pankratz, Dmitry Tochilkin, Christian Laforte, Robin Rombach, and Varun Jampani. Sv3d: Novel multi-view synthesis and 3d generation from a single image using latent video diffusion. *arXiv preprint arXiv:2403.12008*, 2024.

[6] Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang, and Sergey Tulyakov. Infinicity: Infinite-scale city synthesis. *arXiv preprint arXiv:2301.09637*, 2023.

[7] 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.

[8] Haozhe Xie, Zhaoxi Chen, Fangzhou Hong, and Ziwei Liu. Citydreamer: Compositional generative model of unbounded 3d cities. *arXiv preprint arXiv:2309.00610*, 2023.

[9] Mengqi Zhou, Jun Hou, Chuanchen Luo, Yuxi Wang, Zhaoxiang Zhang, and Junran Peng. Scenex: Procedural controllable large-scale scene generation via large-language models. *arXiv preprint arXiv:2403.15698*, 2024.

[10] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *European conference on computer vision*, pages 106–122. Springer, 2022.

[11] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12922–12931, 2022.

[12] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022.

[13] 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. *Communications of the ACM*, 65(1):99–106, 2021.

[14] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models. *arXiv preprint arXiv:2310.12945*, 2023.

[15] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A. Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scene as blender code, 2024.

[16] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[17] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[18] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023.

[19] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[20] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[21] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[22] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*, 2023.

[23] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *arXiv preprint arXiv:2305.19308*, 2023.

[24] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[25] Krzysztof Czerwinski. Autogpt. `https://github.com/Significant-Gravitas/AutoGPT`, 2023. v0.5.1.

[26] Yohei Nakajima. Babyagi. `https://github.com/yoheinakajima/babyagi`, 2023.

[27] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. May 2023.

[28] Yilun Du, Shuang Li, Antonio Torralba, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate.

[29] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, AhmedHassan Awadallah, RyenW White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. Oct 2023.

[30] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2021.

[31] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3d neural rendering of minecraft worlds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14072–14082, 2021.

[32] Arun Mallya, Ting-Chun Wang, Karan Sapra, and Ming-Yu Liu. *World-Consistent Video-to-Video Synthesis*, page 359–378. Jan 2020.

[33] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019.

[34] Lucy Chai, Richard Tucker, Zhengqi Li, Phillip Isola, and Noah Snavely. Persistent nature: A generative model of unbounded 3d worlds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20863–20874, 2023.

[35] Zhaoxi Chen, Guangcong Wang, and Ziwei Liu. Scenedreamer: Unbounded 3d scene generation from 2d image collections. *arXiv preprint arXiv:2302.01330*, 2023.

[36] Cristina Gasch, José Sotoca, Miguel Chover, Inmaculada Remolar, and Cristina Rebollo. Procedural modeling of plant ecosystems maximizing vegetation cover. *Multimedia Tools and Applications*, 81, 05 2022.

[37] Jian Zhang, Chang-bo Wang, Hong Qin, Yi Chen, and Yan Gao. Procedural modeling of rivers from single image toward natural scene production. *The Visual Computer*, 35, 02 2019.

[38] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. Interactive modeling of city layouts using layers of procedural content. In *Computer Graphics Forum*, volume 30, pages 345–354. Wiley Online Library, 2011.

[39] Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Mech, and Vladlen Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2):11–1, 2011.

[40] Carlos Vanegas, Tom Kelly, Basil Weber, Jan Halatsch, Daniel Aliaga, and Pascal Müller. Procedural generation of parcels in urban modeling. *Computer Graphics Forum*, 31:681–690, 05 2012.

[41] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2013)*, 32:Article No. xx, 2013.

[42] Yoav Parish and Pascal Müller. Procedural modeling of cities. volume 2001, pages 301–308, 08 2001.

[43] OpenAI. Gpt-4v(ision) system card. System Card, 2023. Version 1.0.

[44] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[45] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[46] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[47] Google DeepMind Gemma Team. Gemma: Open models based on gemini research and technology. See Contributions and Acknowledgments section for full author list. Please send correspondence to gemma-1-report@google.com.

[48] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[49] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher

Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizingi, Shantanu Jain, Shawn Jain, et al. Gpt-4 technical report, 2023.