Toward a Unified Spatial Interface for Controlling Procedural Content Generation

Achim Bunke
Technical University of Munich
Munich, Germany
achim.bunke@tum.de

Daniel Dyrda
Technical University of Munich
Munich, Germany
daniel.dyrda@tum.de

***Abstract*—Procedural Content Generation (PCG) is central to modern game development, where structured data input and controllability are critical to achieving high-quality, diverse, and context-aware results. This paper proposes a unified spatial interface for controlling PCG based on the Space Foundation System (SFS). By extending the SFS location graph with hierarchical organization and semantic attributes, we provide a shared spatial interface for diverse PCG systems. To integrate existing algorithms, we propose Data Transformers—modular components that translate location graph data into algorithm-specific parameters. We demonstrate the framework's applicability through three implemented scenarios in the context of asset placement for environmental generation. Our results highlight the potential of the SFS to streamline PCG workflows, enhance modularity, and support both automated and designer-driven content creation. This work contributes a step toward a unified spatial interface for PCG in games.**

***Index Terms*—Game Engineering, Game Design, Procedural Content Generation, Space.**

I. Introduction

Procedural Content Generation (PCG) techniques have become a powerful asset in modern game development, enabling the efficient production of rich, varied, and scalable game environments while reducing the manual workload of designers. The application of PCG often balances the effort of designing input parameters and the predictability of the output [1]. More configuration options give designers greater control over the generated results. This is important when having a clear vision for the result, reducing the time required for testing and iterating. A major challenge is the lack of unified control interfaces: PCG algorithms are typically bound to tool-specific or engine-specific implementations, hindering interoperability and complicating multi-system coordination.

A. Our Approach

In this paper, we propose an approach to integrate and control PCG systems through the Space Foundation System (SFS) [2]. By standardizing how spatial data is represented and accessed, SFS enables consistent interaction across heterogeneous PCG algorithms, improving the coordination, coherence, and reusability of PCG systems in game environments.

The SFS formalizes space as a graph of *locations*, where each location is defined by *insideness*—e.g., being *inside* the city or being *inside* a room. Drawing on the notion of space signatures [3], the approach divides the continuous game space into discrete segments and models their relation in a *location graph*. This graph underpins a location-aware API in the game engine, augmenting traditional *Transform* components. This allows any object to query and subscribe to its location in realtime, eliminating the need for ad-hoc trigger volumes and collision checks. By shifting from raw coordinates to semantic locations, the system streamlines the implementation of spatial rules, aesthetics, and narrative events. [2]

A common scenario in 3D game development is the blockout phase, where designers use primitive shapes to define layout, structure, and gameplay flow. In our approach, the SFS is integrated at this stage by placing anchor elements (e.g., city gates, room centers) and delimiter elements (e.g., walls, terrain edges), automatically generating a location graph that captures the world's spatial structure. As the blockout evolves into a detailed environment, PCG algorithms can be applied using the location graph to ensure spatially consistent, context-aware content. For example, an interior furnishing system can target indoor spaces, while a terrain decorator operates on outdoor regions, both coordinated through a unified spatial interface enabling multiple PCG systems to work in parallel.

B. Related Work

The topic of controllability and structured data as input for PCG is a much-discussed topic in PCG literature. Hendrikx *et al.* [4] provide a comprehensive survey of PCG for games, introducing a six-layered taxonomy of game content and highlighting challenges that motivate the need for more unified and automated generation frameworks. Shaker [5] defines key properties of PCG algorithms, enabling qualitative evaluation of tool usability and generation outcomes. These properties are *Speed*, *Reliability*, *Controllability*, *Expressiveness & Diversity*, and *Creativity & Believability*. Controllability refers to the degree to which PCG can be configured and the predictable influence over the generated results. Togelius *et al.* [1] discuss PCG in the context of balancing the effort of designing input parameters and the predictability of the output.

PCG algorithms that work with a spatial context include Unreal Engine's *PCG Framework*, which includes a *PCG Biome* [6] generating game environments. For this approach, a manual construction of volumes defining the generation

Short Paper

bounds is required. Van der Linden *et al.* [7] present different dungeon generation techniques and note a change in attention from algorithmic refinement and results toward the integration of a human designer and precise algorithm control. Smith *et al.* [8] present a mixed-initiative approach for level design of 2D platformers using pacing-based input parameters. Den Kelder [9] describes a *functional game space model* as an architectural concept for gamespace modeling, focusing on analytical analysis. This model introduces space partitioning based on functional properties to enable automated generation.

Carli *et al.* [10] classify PCG techniques as either assisted or non-assisted based on the level of human effort required. They link the amount of designer input and control to the predictability of results while also acknowledging scenarios where non-assisted generation is advantageous. Moreover, they observe a growing trend toward hybrid approaches that combine both methods for enhanced control—an approach our proposed framework also seeks to support.

C. Identified Gap and Goal

Despite the increasing use of PCG in game development, most systems rely on fragmented, tool-specific spatial representations, such as manually defined volumes or ad hoc coordinate logic. This fragmentation complicates the coordination of multiple generators, reduces modularity, and increases the workload for designers. Existing frameworks rarely offer a unified, semantic spatial interface that enables consistent control across diverse PCG systems.

This paper addresses this gap by proposing the SFS location graph as a generalized interface for location-aware PCG. We present a conceptual framework grounded in prior research and game engine architecture. We demonstrate its potential through concrete implementations, showing how different PCG algorithms can operate cohesively under the SFS model, enabling greater interoperability, modularity, and spatial controllability.

II. PCG Framework

We build on the SFS location graph [2] and extend the concept of location by introducing two extensions: *attributes* and *hierarchical structure*. These enhancements support a unified and semantically rich interface for spatial content generation. The extended SFS is then used as general input by PCG algorithms to generate aspects of the game environment depending on the current spatial context. For this, we propose a framework consisting of three core parts: the *Location Graph*, the *Data Transformer*, and the *PCG Algorithms*, which include classical algorithms and novel algorithms. Figure 1 illustrates the structure and data flow of the proposed framework.

A. Location Graph

1) Attributes: Attributes encode logical, aesthetic, and gameplay-relevant properties of a location, turning spatial regions into meaningful content descriptors. While the graph defines the world's layout, attributes define its substance. For instance, a medieval village may consist of a wall (with attributes such as *DefenseValue*, *Material*) and residential

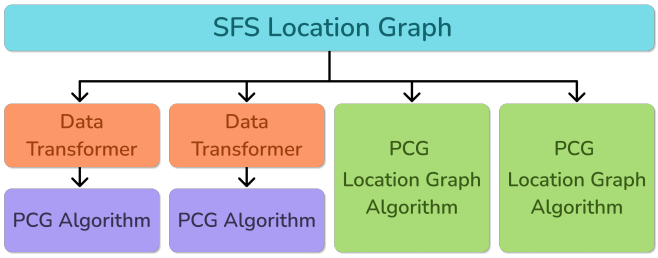

Fig. 1. Schema of the proposed system architecture.

zones (e.g., *Culture*, *Density*). Individual houses can carry specific values such as *Importance*, *Height*, or *OccupantNPC*. These attributes remain relevant across development stages—from early blockouts to final polish—providing a consistent and evolving data model that underpins spatial design. Its structure and contents are iterated in tandem with the visual design of game worlds, enabling PCG integration.

2) Hierarchy: To manage complexity and support abstraction, we extend the graph with a hierarchy that encodes *contains* and *part-of* relationships. This structure groups related locations into higher-level nodes, capturing spatial order through parent-child relationships. Inspired by concepts such as Harel's higraphs [11] and Unity's prefab system [12], this hierarchy supports attribute inheritance with local overrides. A subspace inherits attributes from an enclosing, higher-level node. Designers gain a scalable and intuitive interface, allowing high-level design decisions to influence fine-grained content generation without redundancy. For example, a city node may encapsulate buildings and infrastructure, propagating shared attributes (e.g., architectural style) to its children. A city's design aspects propagate to all subspaces, offering a single design interface compared to hundreds of distinct buildings. These attributes can be overwritten or removed from the location node of a subspace for localized variations.

B. PCG Algorithm

Our framework supports two types of PCG algorithm integration: (1) existing, standalone PCG algorithms adapted via a *Data Transformer* and (2) novel algorithms designed to operate directly on the location graph structure.

1) Data Transformer: A *Data Transformer* is a middleware component that maps spatial data to the input parameters of existing PCG systems. It is tailored to a specific PCG algorithm and processes the location graph, identifies relevant nodes within the target spatial bounds, and translates attributes into algorithm-specific parameters, enabling a more intuitive design process. This may involve simple attribute extraction or more complex semantic decomposition. A simple location description such as *Desert* might need to be split into different PCG parameters such as *Erosion*, *Roughness*, or *Temperature*.

2) PCG Algorithm With Data Transformer: Instead of manually configuring inputs—often a complex and error-prone task requiring algorithm-specific expertise—designers define locations and attributes within the graph. However, many PCG

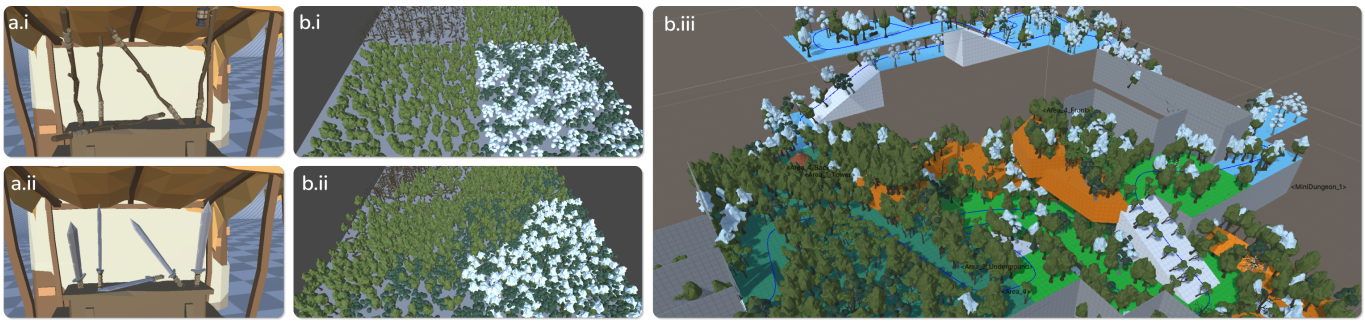

Fig. 2. Examples from the case studies. (a) shows two adaptations of the placeholder tool. (b) shows the results of the asset placer, with (b.i) showing the result of the generic asset placer with clear borders and (b.ii) and (b.iii) showing the results of the location-aware asset placer with smooth transitions.

algorithms were not designed with semantic spatial models in mind and cannot natively interpret the structure of a location graph, thus requiring the Data Transformer to bridge this gap.

3) PCG Location Graph Algorithm: PCG algorithms can be designed to operate directly on the location graph itself. Instead of relying on *Data Transformers* to interpret the graph, these *graph-native* algorithms leverage the hierarchy, spatial relationships, and attributes embedded in the graph as part of their core logic, eliminating the need for translation layers. By embedding spatial awareness into the algorithm, this approach allows for rich interactions with the environment. Structural relationships and inherited attributes can guide procedural decisions, potentially leading to higher-quality and more coherent results compared to solutions adapted in retrospect.

III. The System in Practice

We demonstrate the practical utility of the proposed framework through three implemented case studies. Each case uses the location graph as a unified interface for PCG, showcasing both native integration and adaptation via *Data Transformers*. The cases focus on asset placement during the development process, from initial blockouts to detailed scene construction.

A. Asset Placeholder

This case focuses on the procedural placement of items in market stalls across different cities. Each city contains item stores—modeled as market stalls—that visually display purchasable items. During early development, neither the specific item types nor their distribution across cities is known, and many assets are still in production. To address this, a PCG algorithm assigns and places items based on spatial context. An example result of the algorithm for a weapon store is illustrated in Figure 2-a. Each stall contains predefined placeholders that are populated with concrete item assets during generation. The PCG algorithm requires two inputs: a target stall and a descriptor that determines the category of items to display. Traditionally, this configuration would require manual setup for each stall in every city. In our framework, configuration is automated via the location graph. Each city node is assigned a *ItemType* attribute, which encodes the desired item category. A *Data Transformer* reads this attribute and converts it into the required descriptor. Valid item assets are pulled from a shared asset database. Because the location graph supports hierarchical attributes, item types can be defined at different abstraction levels—such as continent, country, city, or individual stall. This allows default behaviors with localized overrides. Adding new models or changing item distributions becomes a matter of updating location attributes, instantly propagating changes across all relevant stalls.

B. Asset Placer

Procedural placement of natural elements—such as trees, rocks, and plants—is commonly used to reduce manual workload in large environments. This generation depends on the spatial context, as different biomes often require distinct asset sets and visual styles. We demonstrate two approaches for biome-based asset placement: a generic PCG implementation adapted from Unreal's *PCG Biome Core* and a custom location graph–aware generator. The goal is to generate vegetation and environmental features across a continent with diverse biomes. An example result of the various algorithms for simple blockouts is illustrated in Figure 2-b.

1) Generic Asset Placer: This algorithm uses existing asset placer algorithms to populate regions. We assign one PCG instance per biome, and a *Data Transformer* extracts each biome's spatial bounds from its location node, along with attributes such as *Density* or *RotateAssets*, and the asset sets. The result can be seen in Figure 2-b.i.

2) Location-Aware Asset Placer: The location-aware generator is created with the location graph as dedicated input and operates over complex, nested spatial hierarchies. Starting from a root location (e.g., a country), it traverses the location graph to identify subregions and their specific attributes. Position-based asset selection and placement parameters are dynamically adjusted according to the current location affiliation derived from the SFS. This enables consistent generation across biome boundaries and allows smooth transitions between adjacent biomes by leveraging the graph's space adjacency data, as seen in Figures 2-b.ii and b.iii.

C. Asset Recommender System

Beyond autonomous generation, the SFS also supports mixed-initiative PCG tools that aid designers during devel-

opment. We implemented an asset recommender system that assists in interior design tasks, such as populating a living room with small decorative or functional items (e.g., plants, books, or tools) after a coarse blockout of larger furniture.

Traditionally, locating relevant assets within nested folder structures can be time-consuming. Our recommender system addresses this by predicting contextually relevant assets based on the designer's current working location in the engine editor. The system requires asset-specific metadata encoded as descriptive attributes. A *Data Transformer* derives a set of search parameters from the current location node in the graph, using attributes such as *RoomType*, *Function*, or *Style*. These parameters are then matched against the asset database. An integrated editor plugin displays a flat, filterable list of recommended assets, streamlining asset selection during level design. While this system does not generate content directly, it enhances workflow efficiency and demonstrates how the SFS can also facilitate intelligent design support tools.

IV. Discussion

Looking at the resulting proposal, several challenges and limitations must be acknowledged. Constructing and maintaining a detailed location graph with hierarchical attributes can become labor-intensive as world complexity increases. While hierarchy reduces redundancy by allowing attribute inheritance, unique locations still require individual specifications. Although this upfront effort is significant, we expect long-term efficiency gains through centralized control and reduced redundancy across systems. Future work could explore semi-automated tools for attribute generation and maintenance.

The *Data Transformer* provides a bridge between existing PCG algorithms and the location graph, but each new algorithm may introduce novel requirements or attribute types. This creates a growing need to standardize attribute definitions or support extensibility in a controlled manner to preserve interoperability. The complexity of a *Data Transformer* strongly depends on the target PCG algorithm. Simple generators, such as the item stall example, require minimal processing. However, algorithms with more sophisticated inputs—such as terrain generators—may require extensive graph analysis, aggregation, and transformation logic. Developing these transformers necessitates expertise in both PCG and the semantics of the location graph. Many PCG tools intentionally simplify their input interfaces to enhance usability. While this improves accessibility, it can limit how much of the location graph's data can be meaningfully leveraged. In contrast, graph-native PCG algorithms can utilize spatial data more effectively without relying on intermediary translation layers.

Despite certain challenges and limitations, the proposed framework highlights several promising opportunities and implications for research and development. The quality of the spatial information encoded in the location graph directly affects generation quality. Rich spatial metadata can enhance the contextual awareness of PCG systems. This opens opportunities for improving existing generation techniques through better spatial separation and annotation. While our examples focus on asset placement, the SFS-based approach is applicable across a wider range of PCG scenarios. As a generic interface combined with the option for *Data Transformers*, it can support diverse spatial PCG approaches. Tools for mixed-initiative approaches, such as asset recommenders, can benefit from a shared semantic model, improving workflow coherence.

PCG systems often create new spatial subdivisions as part of their procedural creation. These can be reflected back into the location graph, enabling downstream systems to operate with updated spatial information. Supporting dynamic graph augmentation would allow for recursive PCG workflows where generated output feeds into further procedural steps.

V. Conclusion & Future Work

We presented a framework that employs the SFS location graph as a unified interface for PCG. By introducing the concept of *Data Transformers*, we enable location-aware parametrization of generic PCG algorithms without requiring additional configuration. This abstraction facilitates more modular and consistent content generation across diverse systems.

Our examples—ranging from asset placement to mixed-initiative tools—demonstrate the system's versatility in environment generation. However, the framework is not limited to this domain. We see potential applications in areas such as mission generation, dialogue structuring, and runtime procedural systems, which we plan to explore in future work.

The proposed system represents a step toward a unified spatial interface for PCG in digital games. We invite the community to extend the approach with additional *Data Transformers* and PCG modules to further evolve this direction.

References

[1] J. Togelius *et al.*, "Procedural Content Generation: Goals, Challenges and Actionable Steps," in *Artif. and Comput. Intell. in Games*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013, vol. 6, pp. 61–75.

[2] D. Dyrda and C. Belloni, "Space foundation system: An approach to spatial problems in games," in *IEEE Conf. on Games*, 2024.

[3] D. Arribas-Bel and M. Fleischmann, "Spatial signatures - understanding (urban) spaces through form and function," in *Habitat International*, vol. 128, 102641, 2022.

[4] M. Hendrikx *et al.*, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, pp. 1–22, 2013.

[5] N. Shaker, *Procedural content generation in games*, ser. Computational synthesis and creative systems. Cham, Switzerland: Springer, [2016].

[6] Epic Games, "Pcg biome," accessed: May 20, 2025. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-content-generation-pcg-biome-core-and-sample-plugins-in-unreal-engine

[7] R. van der Linden *et al.*, "Procedural generation of dungeons," *IEEE Trans. on Comp. Int. and AI in Games*, vol. 6, no. 1, pp. 78–89, 2014.

[8] G. Smith *et al.*, "Tanagra: a mixed-initiative level design tool," in *Proc. of the 5th Int. Conf. on the Found. of Dig. Games*. ACM, 2010.

[9] R. O. Den Kelder, "Towards a framework for analytical game space design," Master's thesis, Universiteit Utrecht, 2012.

[10] D. Carli *et al.*, "A survey of procedural content generation techniques suitable to game development," in *2011 Brazilian Symposium on Games and Digital Entertainment*, 2011, pp. 26–35.

[11] D. Harel, "On visual formalisms," *Communications of the ACM*, vol. 31, no. 5, p. 514–530, may 1988.

[12] Unity technologies, "Prefabs," 2024, accessed: July 30, 2024. [Online]. Available: https://docs.unity3d.com/Manual/Prefabs.html