Reform Procedural City Generation Using an Extended L-System

Naveen Raj B[1], Santhosh K[2], Umamaheswararao S[3]
[1,2,3]*Department of Computer Science and Design Engineering, Rajalakshmi.Engineering College, Thandalam - 602105. Chennai, Tamilnadu, India*

***Abstract*—Reform - A layer-based procedural generation framework of Reform provides developers with tools that increase their development speed while maintaining accurate navigation through open-world environments. The Unity Engine-based tool creates a separation between visual elements and logical graph information. The pipeline creates road networks that exist in multiple levels while maintaining strict rules about their physical layout. The topological validation system maintains connections between all minor streets and the main network which results in complete accessibility for AI pathfinding. Designers can use the asset-agnostic system to interchange meshes and prefabs at any time. The system creates extensive cities with different visual styles which function as authentic navigational paths within one minute of operation.**

***Index Terms*—Procedural Generation, Urban Planning, Graph Theory, L-Systems, Game Development, Unity Engine, Network Connectivity, Asset Management.**

I. INTRODUCTION

The technique of procedural content generation has become an essential method for creating vast and varied game worlds which require no manual design work in contemporary video game development and virtual environment simulation. Urban area development requires specialized solutions from different procedural methods because designers must balance three different requirements, which include creating realistic geometric shapes and enabling people to walk through the area while allowing creative artistic expression. The traditional methods use fixed grid systems together with simple L-system designs which generate repetitive patterns, create broken road systems, and produce topological defects that damage both gameplay and artificial intelligence pathfinding abilities.

The primary challenge in procedural city generation lies in balancing three critical requirements: computational efficiency, structural connectivity, and asset flexibility. Existing systems generate road networks through fast methods but create problems because they do not connect all streets to the primary network which results in unreachable road sections that disrupt pathfinding operations.

The research introduces REFORM as a hierarchical system which uses graph-based procedural generation to solve existing system limitations. The developed system uses an advanced L-system method which enables it to create urban design patterns that include major arterial roads and minor local streets while supporting different urban density levels and naturalistic street design elements. The system applies graph traversal methods to find and remove road segments which do not connect to the network while maintaining emergency street access and removing inaccessible areas that disrupt gameplay.

II. SYSTEM DESIGN

2.1 Overall Architecture

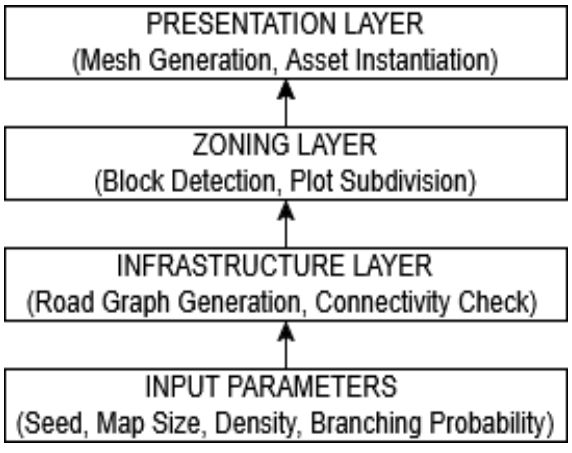

Figure 2.1: System Architecture Overview

The system follows a hierarchical pipeline structure, as illustrated in Fig.2.1. The architecture consists of three layers that operate in sequential order. Infrastructure Layer: The core road network graph is created through an extended L-system algorithm. Zoning Layer: The road graph is analyzed to determine which areas can be developed and which areas will remain vegetated. Presentation Layer: The layer creates visual elements through the semantic information obtained from earlier layers. The process of visual asset changes requires no city structure regeneration because the system maintains separate visual assets and city structure components, which leads to faster development iterations.

2.2 Data Model

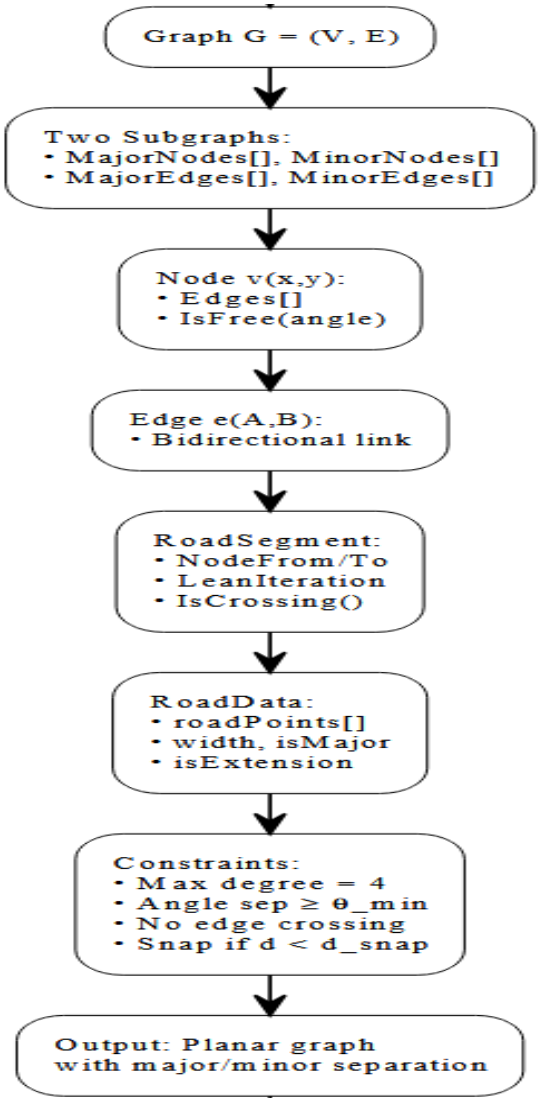

Figure 2.2: Graph Data Structure Representation.

A planar graph serves as the primary data structure which uses intersections as vertex points and street segments as edge connections. The graph maintains two disjoint sets to distinguish between arterial and local streets: Node Structure: Each node stores spatial coordinates and maintains a list of connected edges. Nodes use angular restrictions to control the density of their intersection points. A node is considered valid for new connections only if the angular difference between existing edges exceeds a minimum threshold. Edge Structure: Each edge connects two nodes and stores metadata including road type (major/minor), width, and extension status. Edges are bidirectional and support traversal in both directions for pathfinding and validation algorithms. Road Segment Data: Road segments store waypoint sequences for curved roads. The segment includes a control point list which it maintains.

2.3 Generation Engine

The generation engine applies the extended context-sensitive L-system through two dedicated modules which handle arterial and local street networks. The arterial module starts with a random seed point which lies within the map boundaries and it creates growth segments that expand in both directions through repeated growth cycles. The processing queue controls the active segment front, which expands in an organized way until it reaches its maximum segment limit. The production rules control segment extension together with probabilistic branch creation and curvature changes during each iteration. The system uses curvature control through an iteration counter which tracks segment rotation, creating smooth curves that replace sharp corner angles. Branching occurs probabilistically based on a configured density parameter, spawning perpendicular segments to form intersections. The local street module operates sequentially after arterial generation, spawning initial segments perpendicular to existing arterial nodes. This module uses its strong bidirectional branching rules to create grid patterns that define residential block areas. Both modules use a complete constraint system which checks each new segment against geometric and topological regulations. The system uses three different types of boundary checks which include map boundary constraints and line-intersection tests and proximity snapping for intersection validation. The system uses node degree limits to restrict intersections

to four edges, which follows typical urban planning practices. The refinement process, which starts after the growth phase, eliminates all isolated segments while enforcing all required connections to create a structured graph which contains valid nodes and edges that will undergo topological validation and mesh creation.

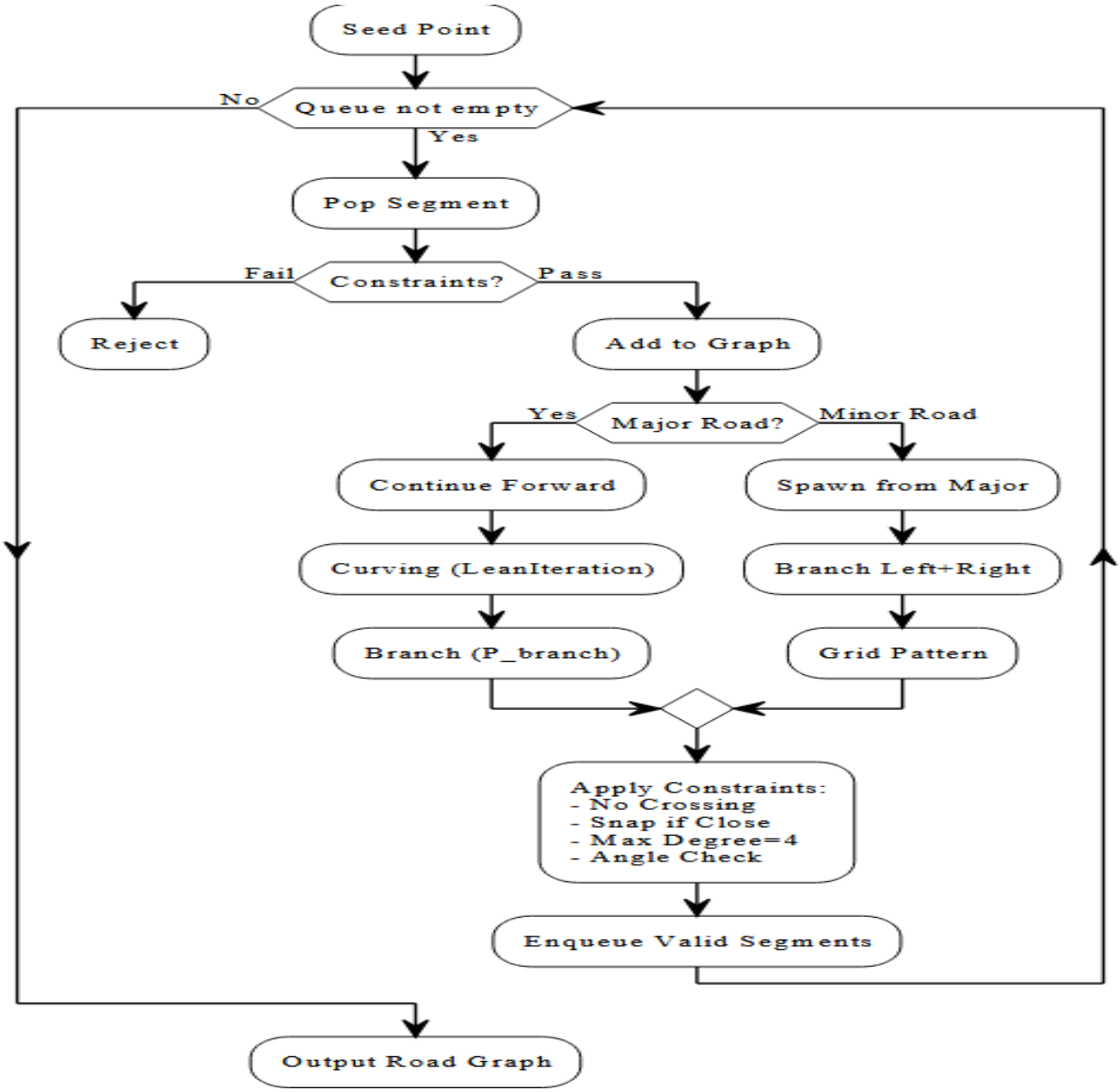

Figure 2.3: :Extended L-System Production Rules

2.4 Validation System

The system performs comprehensive validation after the first-generation step to verify both topological integrity and navigational accuracy. The post-processing module uses Breadth-First Search (BFS) algorithm to check graph connectivity which solves the standard problem of unreachable road segments in procedural generation. The process starts when all minor nodes that connect to the arterial major network are identified because they serve as starting points for node traversal. The algorithm explores adjacent minor nodes to discover all components that link to the local street subgraph. The graph traversal process must visit all minor nodes and edges because any unvisited elements will be considered disconnected and removed from the graph. The additional cleanup passes delete all isolated edges and all orphan nodes that lack connections and all internal dead-ends which

prevent loop creation. The validation process verifies that all generated streets maintain direct access to main network pathways while removing any street segments that would create obstacles for artificial intelligence and player navigation. The road network achieves complete connectivity which allows asset implementation to proceed without any need for manual adjustments.

III. IMPLEMENTATION

3.1 Development Environment

The core generation modules were developed as pure C# classes independent of Unity's Nonbehavioral lifecycle, allowing headless execution for performance testing. The graph data structures use generic collections together with hash-based lookups to achieve O(1) average-case complexity when performing node and edge operations. All random number generation uses seeded System. Random instances which ensure reproducible results that can be used for debugging and comparative analysis.

3.2 Layered Execution Pipeline

The system starts its operations by executing the infrastructure layer which processes the extended L-system rules through its iterative queue-based expansion algorithm. The evaluation of each segment uses geometric restrictions which include tests for line intersections and checks for angular separation and methods to validate boundaries. The validation layer executes a breadth-first traversal process to establish connectivity while it removes disconnected parts of the graph at a linear time rate based on the size of the graph. The presentation layer transforms semantic road information into its displayed form.

3.3 Performance Optimization

The environment needs multiple implementation approaches to create successful results which complete their tasks within a one-minute time frame for big urban areas. The initiation of mesh generation needs to wait until all graph operations complete because this approach reduces Unity engine processing burden during logical operations. The system marks generated meshes as dynamic to enhance the performance of vertex buffer updates. The system uses coordinate-based hashing to monitor node uniqueness which helps stop duplicate processing from happening. The generation loop uses early termination implementation because it stops execution when segment limit values reach their maximum thresholds, which prevents unlimited system expansion. Profiling shows that graph construction takes about 65 percent of total generation time while mesh instantiation uses the remaining 35 percent.

IV. FUTURE ENHANCEMENT

4.1 Parcel Generation and Street Lighting

The system proceeds to parcel generation by analyzing closed loops within the validated road graph. The algorithm identifies city blocks by traversing edge cycles and computes interior boundaries using polygon clipping techniques. The system generates individual parcels from each block according to zoning rules which define minimum lot width and depth variation and setback requirements. The road graph serves as the foundation for street lighting which functions as an additional layer that operates in parallel. Street lights are installed at specified distances along the centerlines of roads, with different spacing requirements established for each road classification. The logic for placement determines intersection geometry through its main focus on corner positions while it restricts placement to areas outside of node circle boundaries.

4.2 Dynamic Traffic Simulation and AI Navigation

The created graph structure of the generated output functions as a complete pathfinding solution. The system will achieve its next development stage through the implementation of direct navigation mesh data integration which will occur during road construction processes for instant AI agent activation. The system will implement a lightweight traffic simulation module which will use edge metadata to create models of vehicle movements and traffic congestion and real-time routing changes that support emergency response and logistics management gameplay elements.

4.3 Multi-Terrain Adaptation and Heightmap Integration

This system operates on a flat plane. The system needs heightmap support so it can generate cities that exist on multiple types of terrain which includes hills valleys and coastlines. Road segments could adapt

their elevation and curvature to follow terrain contours, while validation constraints would prevent unrealistic slopes or intersections. The framework can be used in multiple game worlds because this development expands its functionality.

V. CONCLUSION

This research presents the design and implementation of "REFORM", a procedural city generation framework built on Unity Engine. The project successfully creates an urban environment which developers can populate with large worlds through its organic scalable design which enables efficient navigation of the created space. The Graph-Based L-System provides a validated platform for reliable infrastructure generation which uses different traditional procedural methods that create disconnected road networks which break AI pathfinding. The system achieves high performance because it separates the generation logic from the validation system and visual representation modules which operate independently. The "Topological Validation" logic flow prototype demonstrates its capability to execute complex graph algorithms in operational workflows. Major roads create natural network structures while minor roads provide essential access throughout residential areas.

The research demonstrates that creators can build their projects through procedural generation because it offers developers a scalable and efficient development platform.

VI. REFERENCES

[1] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in Proc. ACM SIGGRAPH, 2001, pp. 301–308.

[2] M. Kelly and S. McCabe, "A survey of procedural techniques for city generation," ITB Journal, vol. 7, no. 2, pp. 25–37, 2006.

[3] E. Galin, E. Guérin, A. Peytavie, P. Cordonnier, B. Benes, and M. Brisson, "A review of procedural modeling methods for virtual worlds," Computer Graphics Forum, vol. 39, no. 3, pp. 811–837, Jun. 2020.

[4] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modelling for virtual worlds," Computer Graphics Forum, vol. 33, no. 6, pp. 31–50, Sep. 2014.

[5] S. A. Groenewegen, R. M. Smelik, K. J. de Kraker, and R. Bidarra, "Procedural city layout generation based on urban land use models," in Proc. Eurographics, 2009, pp. 49–58.

[6] E. Galin, A. Peytavie, N. Wray, and R. Gaillard, "Procedural generation of roads," Computer Graphics Forum, vol. 29, no. 2, pp. 429–438, May 2010.

[7] E. Galin, A. Peytavie, E. Guérin, and B. Benes, "Authoring hierarchical road networks," Computer Graphics Forum, vol. 30, no. 7, pp. 2021–2030, Sep. 2011.

[8] M. Hendrikx, S. Denies, F. De Troyer, and J. Denil, "Procedural content generation for games: A survey," ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 9, no. 1s, pp. 1–22, Feb. 2013.

[9] A. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "Integrating procedural generation and manual editing of virtual worlds," in Proc. IEEE Conf. Comput. Intell. Games, Aug. 2010, pp. 311–318.

[10] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," ACM Transactions on Graphics, vol. 22, no. 3, pp. 669–677, Jul. 2003.

[11] B. Lintermann and O. Deussen, "Interactive modeling of plants," IEEE Computer Graphics and Applications, vol. 19, no. 1, pp. 56–65, Jan./Feb. 1999.

[12] A. Lindenmayer, Mathematical Models for Cellular Interactions in Development. New York, NY, USA: Academic Press, 1968.

[13] T. Merrell, "Procedural generation of road networks for driving simulation," in Proc. IEEE Int. Conf. Simul. Model. Pract. Theory, Oct. 2015, pp. 112–119.

[14] J. Sun, X. Yu, G. Baciu, and M. Green, "Template-based generation of road networks for virtual city modeling," in Proc. ACM Symp. Virtual Reality Softw. Technol., 2002, pp. 33–40.

[15] K. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," ACM Transactions on Graphics, vol. 25, no. 3, pp. 614–623, Jul. 2006.