Leif Clark

CS246E Final Project Design Document

Geometric Procedural Cities

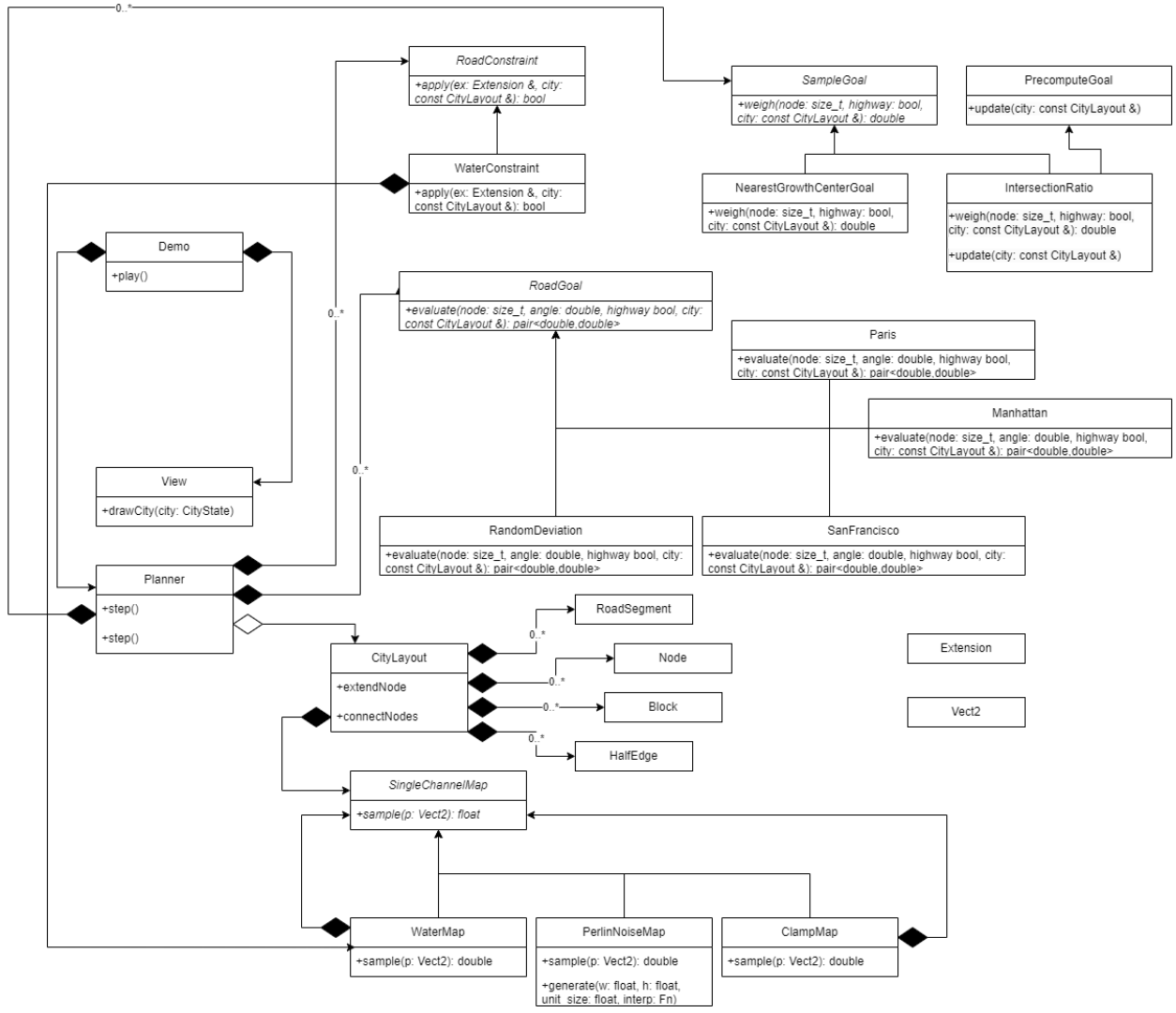
Introduction

The goal of this project is to build a library for procedurally generating the geometry of two dimensional city maps over time. That is, the top down geometry of a city, not restricted by a grid or other simplifications of two dimensional space, and capable of dynamically expanding over time. The project was planned with game development in mind. The geometric approach allows the city map to be used for a wide variety of applications. Most notably, geometry generated from this library lends itself well to future levels of procedural detail including vertical expansion into three dimensions. For example the geometry outlining a road intersection or an apartment building could be fed to a potentially 3D procedural intersection or apartment generator. Furthermore, the ability for the city to grow over time creates great potential for the city map generator to interface with game mechanics such as other simulations and player interaction. In the "Half-Life: 25th Anniversary Documentary," Gabe Newell explains that when designing Half-Life, part of Valve's definition of "fun" was a reactive environment that acknowledges player actions. A dynamic procedural city would be a wonderful environment for many games from tycoons to RPGs. Finally, an efficient implementation of this library will allow cities to be generated quickly enough to be used in real-time in a game. The game-related motivation of this project is important to note as most city simulations and generators are used for urban planning, where the actual geometry is less important, or for computer graphics in film where often only a final result is required and interactivity is of no interest. In order to make a convincing simulation, a level understanding of urban planning is required. My knowledge of this subject is nonexistent. Thus, I took many procedural techniques from both of the papers "Procedural Modeling of Cities" by Yoav I H Parish and Pascal Müller and "Interactive Geometric Simulation of 4D Cities" by Basil Weber, Pascal Müller, Peter Wonka, and Markus Gross. These papers also provided some advice on computational geometry. Most notably the suggestion to use a half-edge data structure to represent the planar graph created by roads.

Overview

The model consists of two major parts, the planner and the city layout classes. A city layout is a half-edge data structure representing a planar graph where edges are roads, vertices are road intersections, and faces are blocks. City layout objects contain all information about a city at a certain point in time. It has methods to add new roads into the data structure while maintaining the planar graph invariant as well as a minimum angle between roads and minimum road length. The job of the planner class is to propose roads to add to the city layout. It does so in three steps. First it samples an intersection from the city layout. Next it chooses an ideal length and angle to extend a new road from the selected intersection. Finally it applies its own invariants before requesting that the city layout add the proposed road. The planner uses sample goal, extension goal, and road constraint objects to make these decisions. It can apply a variety or even a combination of any of these goals and constraints using the strategy pattern. Sample goals can be used, for example, to prioritize city growth in a certain area or to achieve a target ratio between two different degrees of intersection (Perhaps you want half as many 4-way intersections as 2-way intersections (straight segments)). Extension goals can be used to encourage roads to follow elevation contours, align to an arbitrary grid system, create concentric circles, or follow the gradient of a population map. Road constraints can be used to prevent roads from crossing water (or if they do mark them as bridges), avoid areas of low population, or adhere to a maximum vertical slope. The planner also has two modes. It is either planning highways or secondary streets. It plans highways until it creates an induced cycle of highways. We will refer to this induced cycle as a "quarter." The planner then fills in the quarter with secondary streets. This process ensures that secondary streets do not block or otherwise interfere with the extension of highways. Had I finished the project, the planner would also have similar strategy patterns for subdividing blocks into lots, constructing buildings on lots, and differentiating between planning and actually constructing roads. Classes within these patterns would implement a land use simulation to simulate land use types, economy, and traffic. The project also has a view class that when provided a reference to a city layout will render the city to the screen using SDL.

UML



A noticeable change to the UML is the lack of some features such as buildings, a traffic simulation, and a city timeline. This change is not particularly interesting because were these features implemented they would likely fit the same position on the UML as originally planned. Another two minor changes are the use of doubles instead of floats (I wanted more precision) and the new classes Vect2 and Extension. Vect2 encapsulates and greatly reduces the code required to do vector operations in R^2. Extension is a struct to represent a new road before it has been added to the city layout data structure. I also added a Demo class to separate user input from the view in accordance with the single responsibility principle. The most nuanced changes to the UML are the removal of aggregation relationships from RoadSegments, Nodes (formerly

labeled “Intersection”), and Blocks, the addition of the HalfEdge struct, and the new SampleGoal and PrecomputeGoal system. These modifications involved careful design decisions that are described below.

Design

The most difficult design challenge I faced was defining, encapsulating, and implementing the responsibilities of the CityLayout class. When planning my project I understood that a polygon or edge “soup” method of storing road and block information was insufficient. I soon also realized, however, that a simple graph data structure would not provide the efficiency I desired. Thus, I represented my city with a half edge data structure. A half edge data structure is a type of directed multigraph commonly used in computer graphics to represent meshes. Meshes are similar to planar graphs so this data structure is also ideal for my application. Half edges allow easy traversal around both blocks and nodes making the detection of induced cycles a trivial task as long as invariants are maintained. Maintaining the invariants of a city layout, most importantly the planar graph invariant, forced me to change my initial approach where most responsibilities were given to the planner class. The city layout needed to check for intersections itself when adding a new road to maintain planarity. Thus the IntersectionConstraint was removed from the planner class and this responsibility was delegated to the city layout. This invariant also meant that other classes should not be able to directly edit city features relating to positional information. Thus, the city layout must not return mutable references to roads, nodes, or blocks. However, passing constant road, block, or node references to the city layout when requesting edits to positional data does not allow the city layout to easily make the required changes. Thus, the city layout should primarily interact with other classes through some form of iterator. I chose to just consume and return indices of roads, blocks, and nodes as I desired random access, and since roads are never removed from the city layout, indices are never invalidated. Aggregations were removed from nodes, roads, and blocks as well to further strengthen encapsulation and because it was intuitive for the city to use the same index-based system internally. Another interesting design challenge was the implementation of sample goals. When beginning work on my project I realized that the priority queue / L-system approach used in “Procedural Modeling of Cities” to select new roads for expansion was insufficient to provide a compelling simulation over time, only its final results being useful.

“Interactive Geometric Simulation of 4D Cities” suggested random sampling nodes for expansion. Thus, I needed a system to control the probabilities with which these samples were taken. I implemented a strategy pattern that would assign a weight to each node. Unlike extension goals and road constraints, however, these sample goals for every single node, every single time an expansion is planned. Therefore, a goal such as the IntersectionRatio goal should ideally only calculate the ratio between different node degrees once whenever the city layout is updated. My solution to this issue was the PrecomputeGoal class. Rather than extend the sample goal interface in a way that many sample goals would never use, sample goals can use multiple inheritance from precompute goal to adapt their interface with a simple observer pattern and be notified whenever the planner class edits the city layout. This is also an example of the adaptor pattern.

Final Question and Conclusion

If I had the chance to start this project over I would plan ahead more and start earlier. The changes to invariants and encapsulation of city layout set me back an entire 30 hours of work. If I had spent more time considering the desired features of my project I may have been able to better define responsibilities and invariants for the city layout and avoided a lot of bugs and rewritten code. Going into this assignment I was used to simpler programming languages and was worried that I would agonize over small details too much and get little done. I have since realized that a C++ project of this scale requires far more thoughtful planning than I anticipated due to complexities brought by memory management. I intend to continue my project in the future (possibly even over the break), and will certainly create a larger and more carefully thought out UML for future extensibility. I want to support far more complicated combinations of multiple planning rules, design a more versatile city layout, and maybe even abstract constraints and extension goals into one class.