Procedurally Generated 2D Dungeons using Voronoi Diagrams and Delaunay Triangulations

Aadit Nagori

Abstract

This paper outlines an innovative approach to procedural generation of cave systems or dungeons for 2D video game environments utilizing the CGAL computational geometry library. The methodology involves utilizing Delaunay triangulations and their subsequent Voronoi diagrams to create a semi-realistic cave system. Cells are then selected iteratively from said system to create and visualize the dungeon. This approach allows for the creation of more realistic playable dungeon systems for 2d platformer games rather than dungeon systems populated only by box-shaped rooms and corridors linking them together.

Motivation

I have always liked roguelike – dungeon explorer games and one of my favorite games is a game called HollowKnight which has an extremely expansive and rich dungeon system to explore and play in. However, the dungeon map itself mainly consists of rectangular box shaped rooms connected with each other using long thin corridors. Hence, I wanted to explore the utilization of procedural cave generation to create randomized but more realistic explorable dungeon maps for 2d platformer games.

Related Work

“Procedurally Generated Dungeons”

This article focuses on a more input based approach where the developer creates and places rooms in a grid arbitrarily and the algorithm creates a Delaunay triangulation to firstly connect the rooms together and then creates a minimum spanning tree of the Delaunay triangulation to create the corridors joining the rooms together. I used this article mainly to understand and learn about the cave generation process but I did not use the algorithm outlined in the article.

“Procedural Playable Cave Systems based on Voronoi Diagram and Delaunay Triangulation”

In this paper, (Santamaría-Ibirika et al.) propose and devise a technique similar to the one I decided to implement with some key differences. The algorithm outlined in this paper only utilizes a single Delaunay triangulation of generated seed points, the Voronoi diagram of which is utilized to create the cave system. The paper then uses a varying number of inputs from the user to select the cells in the cave system using probabilistic functions based on those input parameters. This algorithm gives the developer freedom to just create the types of rooms they may want in their dungeon (Ex: treasure room, enemy room, entrance etc.) and the algorithm randomly places them based on the input parameters associated with each type and probabilistic calculations. This paper was of immense help in devising my own approach to tackle the problem of randomized procedurally generated cave systems.

Implementation

The input for the algorithm outlined are the different POI types that the user wants to include in their dungeon. Each POI type has 4 main attributes attached to it – id, location, minimum depth and maximum depth. The list of attributes attached can be expanded to include parameters like cavity size and branches which can further add to the control given to the developer and thus lead to more suitable dungeons specific to the developers needs. However, due to the scope and time constraint of this project I was only able to implement and accommodate for the four aforementioned attributes.

ID	Location	minDepth	maxDepth
0	surface	0	0
1	inside	150	300
2	inside	250	500

Table 1: Example POI Inputs

The implementation process for this algorithm can be split into 4 steps.

1. Seed Point Generation and Primary Triangulation
The first step is the creation of the seed points that will be used for the Delaunay triangulation. The seed points are randomly generated using a white noise distribution pattern in a grid of inputted size. I utilized a white noise distribution pattern as it generated points with randomness while being somewhat spaced out and uniform, ensuring that all of the points weren't congested in a single area of the grid as that would lead to the generation of extremely dense cell distribution in only one part of the map whereas the rest of the playable map would be extremely sparse. Such a dungeon map would not be fun to play in and would thus go against the overall objective of this paper.

The number of seed points to be created are calculated based on this formula from (Santamaría-Ibirika et al.):

$$Seeds = \frac{N}{C}S^3$$

Where N is the number of pixels in the terrain plot where the cave will be generated, C is the underground cavity global size parameter and S is the terrain scale. In my implementation, as I utilized CGAL instead of a game engine, the pixel number parameter did not matter as much and was arbitrarily set to 1000. However, as the algorithm outlined in this paper can be easily converted and implemented in a game engine like Godot or unity, the N parameter will become much more important depending on the size and underlying architecture of the game it is being utilized for. The seed points generated are then triangulated using Delaunay triangulation which is done with the CGAL Delaunay Triangulation adapter.

(a) (b)

Fig 1. (a) Seed Pattern of 1000 Points. (b) Delaunay Triangulation of those seed points

2. Secondary Triangulation and Voronoi Diagram
The next step is to get the centroids of all of the triangles created from the primary Delaunay triangulation and generate a second Delaunay triangulation using those centroid points. While the process can be implemented using only the primary triangulation, in my testing I found that utilizing the secondary triangulation for the Voronoi diagram instead led to a more even spread of the sites and cells in the Voronoi diagram and thus were more equidistant while having randomized shapes. I felt that these more even spread-out cells generally led to more interesting cave system formations and more importantly somewhat prevented the dungeon system created to be majorly skewed in any direction and thus led to a dungeon with better playability for the user. While all of the points utilized are randomly generated and thus ensured that every cave system and dungeon created would be completely different even with the same N,C and S values, I manually added four points to the list of centroids for the secondary triangulation and subsequent Voronoi diagram. The points were at the coordinates (width/2,height*2), (width/2,-height), (width*2,height/2), (-width,height/2). These points had to be added as without them, for smaller cave systems with around n = 100 points the Voronoi diagrams created were not suitable for a cave system and thus led to the creation of unplayable dungeons. Hence by manually adding these four points, the triangulation and Voronoi diagram are more constrained leading to the creation of a suitable cave system in the inner grid.

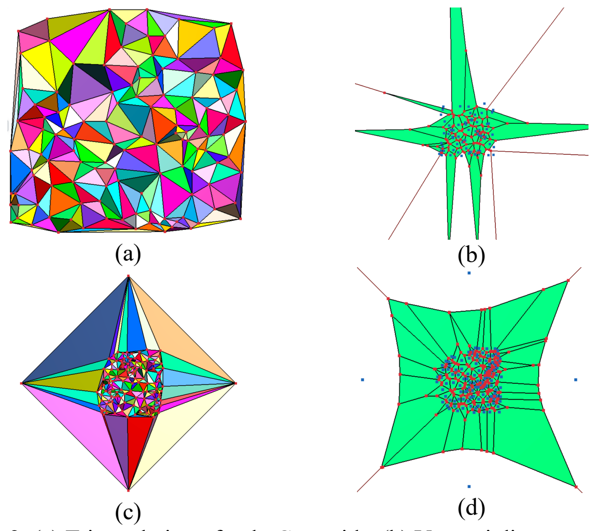
Fig 2. (a) Triangulation of only Centroids, (b) Voronoi diagram of (a), (c) Triangulation of centroids with manual points, (d) Voronoi diagram of (d).

3. Cave System Creation

The next step is to then take the Voronoi diagram generated and remove the external boundary faces from it. As a safety precaution, any faces with vertices greater than or lesser than the height or width set by the user are also removed. The faces remaining after the removal process are then a part of the overall cave system. After this, the surface faces, or the faces which are a part of the surface of the cave system have to be identified. This can be done either by iterating over all of the faces in the cave system and finding the boundary faces with vertices greater than a certain arbitrary number or by selecting all of those faces which have a vertex greater than some arbitrary number. I decided to go with the second option as I wanted to optimize the already expensive process a bit more by avoiding an expensive is_boundary search on every face. The arbitrary constant I found to work best for my needs was height – 55 and in my testing, irrespective of the number of cells, it was consistently able to select only the topmost layer of cells from the cave system and mark them as surface cells.

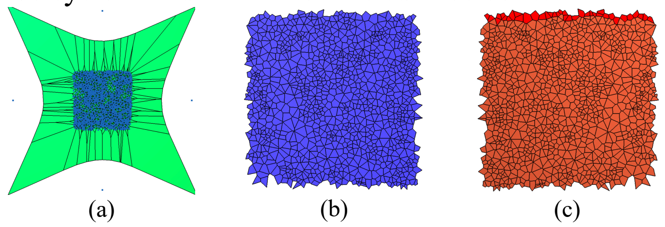
Fig 3. (a) Voronoi diagram, (b) Voronoi without boundary faces, (c) cave system with red = surface and brown = inside

4. Cell Selection and Dungeon Creation

The last step is the selection of the cells from the cave system that will form the dungeon map. This is done by firstly, randomly selecting a cell from the surface cells (preferably from the center which is achieved by finding the faces with a distance of an arbitrary constant from the width/2 value. In my testing I found 10 to be the perfect arbitrary constant. And randomly selecting the surface POIs from them). After this, for every subsequent face starting from the first surface POI, the faces which surround that face and which have an average y index value similar to the current face or those which have an average y index value lower than the current face are selected and the next face is randomly picked from those selected faces. The average y value is calculated by iterating over all of the vertices of that face and averaging the values of their y index with the number of vertices in that face. The similarity between the current face and selected face is determined by ensuring that the difference between the average y values of the two faces is lesser than an arbitrary constant. In my testing 50 was the perfect arbitrary constant as it selected cells that were not only below but also next to or slightly above the current face, leading to more variability in the faces chosen and thus more interesting dungeon patterns. As a design choice, I ensure that for the first five iterations, the face chosen to be the next face has the lowest average y index value from the selected faces. I did this as I wanted to ensure that the dungeon would go deeper and would try to select cells from deeper layers of the cave system and not congregate towards the top. During the iterations, specific cells are marked as POI's randomly, if the current number of iterations is within the depth range inputted by the user. Hence by selecting the next cell with controlled randomness one at a time, the dungeon is iteratively created.

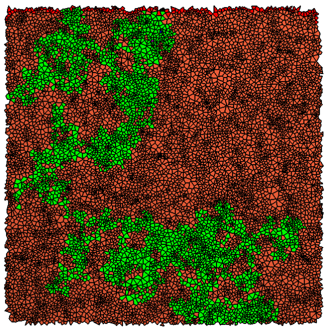

Fig 4. Dungeon generated from cave system having 1000 cells.

Results

The results yielded from this program were quite promising and I was able to achieve what I had set out to do. The patterns and dungeons created by the program are visually extremely interesting and I know I personally would thoroughly enjoy most of the dungeons generated by the program in a video game. Even with the same POI's and input parameters, dungeons generated are almost always extremely different from each other and have an extremely high variability. The program is also surprisingly quick and as the table below shows, is able to generate the cave system and create the dungeon for large inputs fairly quickly.

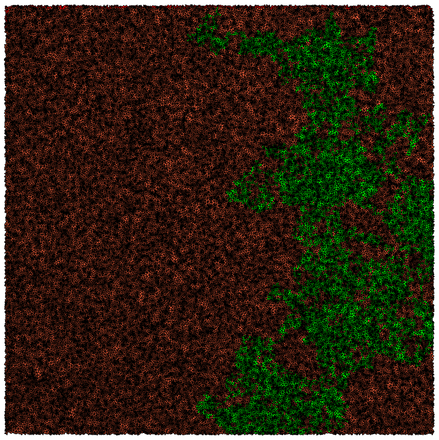

Fig 5. Dungeon generated from cave system having 100000 cells.

Cell No	Cave Time (s)	Dungeon (s)
1000	0.03	0.01
10000	1.8	1.3
100000	14.3	7.4

Table 2. Average times of Cave and Dungeon Creation for varying cell numbers.

Conclusion

This project took around 3 weeks of on and off work and I was able to achieve what I set out to do. The results are exactly what I had hoped to achieve. Currently there is very little research or implementation of such randomized procedurally generated dungeon systems for games in the industry, with majority of developers either choosing strategies similar to (Vazgriz) or ensuring that the cells generated are only in square shapes like Spelunky or Minecraft. However, I feel that if implemented, such Voronoi based dungeon systems would also be extremely interesting dungeon systems that would be extremely enjoyable.

Future Scope and Limitations:

The main limitation of my algorithm is the degree of control given to the developer. As mentioned earlier, I only take in 4 input parameters and only accommodate for a single POI of each type in my current implementation. However, adding more attributes to the POI's or handling multiple POIs per type could be easily implemented and would only lead to changes in part 4 of the overall implementation. Another addition would be allowing the developer to set branches, hence from a single POI or cell, multiple branches could occur leading to even more interesting dungeon systems. To accommodate for branching a recursive approach could be utilized. Finally, another main improvement would be further optimizations of the cell picking algorithm as it is currently at worst case a $O(n^2)$ operation but could be improved significantly.

Works Cited

Adonaac, A. "Procedural Dungeon Generation Algorithm." Game Developer, 3 Sept. 2015, www.gamedeveloper.com/programming/procedural-dungeon-generation-algorithm.Santamaría-Ibirika,

Aitor, et al. Procedural Playable Cave Systems Based on Voronoi Diagram and Delaunay Triangulation. 2014.

van der Linden, Roland, et al. “Procedural Generation of Dungeons.” IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 1, Mar. 2014, pp. 78–89, https://doi.org/10.1109/tciaig.2013.2290371.

Vazgriz. “Procedurally Generated Dungeons – VAZGRIZ.” Vazgriz.com, 2019, vazgriz.com/119/procedurally-generated-dungeons/.