Level of Detail Navigation Algorithm based on Recast

This article implements a multi-level pathfinding method based on the area information within the Recast&Detour algorithm (referred to as Recast), which can effectively reduce the memory usage required for pathfinding while achieving high accuracy and speed. The H42 project has already integrated this solution into the engine, made relevant modifications to the script-level logic, generated data for several scenarios, and released it to the external network. Currently, it is available to users on 32-bit Windows and Android platforms, primarily addressing the substantial memory requirements for large-scene pathfinding on low-performance devices.

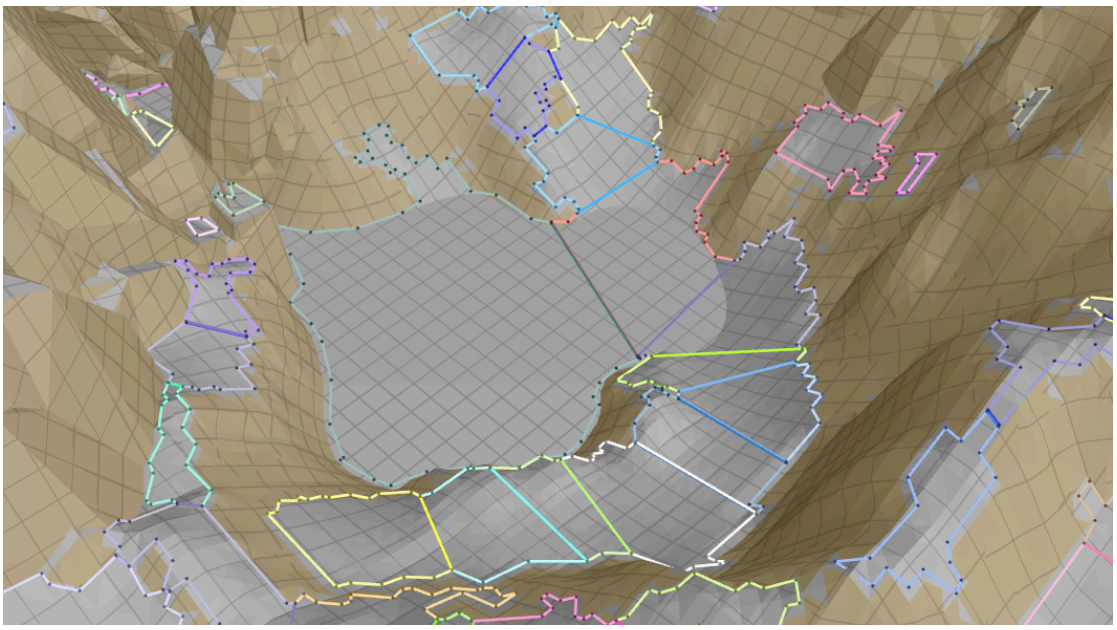
1. Background

The Recast pathfinding algorithm used by Messiah is known for its accuracy and efficiency and is suitable for most game scenarios. However, when used in the "Western Region" (4km x 4km) scene of "One Dream Jianghu," we found that the pathfinding data generated by Recast's algorithm occupied nearly 60MB of memory, and loading pathfinding data when switching scenes also took a long time. Messiah's large-world pathfinding solution is based on a segmented pathfinding map (CAMap) scheme, which can reduce the memory usage of pathfinding data in large maps. However, this scheme requires specific management of the loading and unloading of segmented pathfinding map data as the player moves, and sometimes the actual pathfinding results do not meet expectations. Additionally, the 2016 "Hierarchical Path-Finding for Navigation Meshes" introduced a method based on hierarchical pathfinding, which consumes less memory and has faster pathfinding speeds, but cannot guarantee the same accuracy as the Recast pathfinding algorithm.

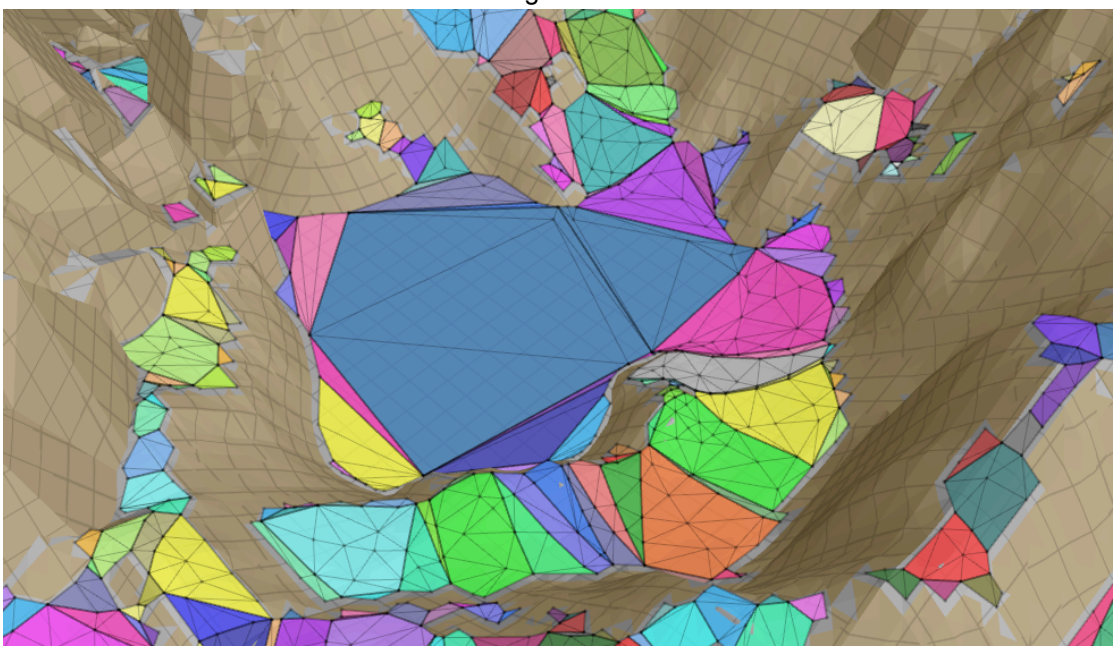
To reduce the memory usage of pathfinding data in large worlds and ensure the accuracy of pathfinding, our approach is to use the intermediate data from the Recast pathfinding algorithm to construct high-level pathfinding data, load all high-level pathfinding data into memory, and store the triangular pathfinding data in files. When needed, we read the corresponding area's triangular data into the cache. Test data shows that the Recast-based pathfinding algorithm requires 61MB of memory and an average time of 0.38ms per pathfind in the Western Region scenario; the multi-level pathfinding scheme (this scheme) based on Recast requires an initial 4.8MB of memory, reaching up to 11MB after multiple pathfindings, with an average time of 3.29ms per pathfind. The starting and ending positions are accurate, and the pathfinding results are generally consistent with the Recast pathfinding algorithm.

2. Technical Approach

This scheme builds upon the data construction process of Recast's pathfinding, using area data, polygon data, and other relevant data to construct both high-level and low-level pathfinding data. This step is called "hierarchical data construction." During specific pathfinding tasks, pathfinding is first performed using the high-level data, which contains fewer numbers and has a simpler structure. Then, based on the results obtained from the high-level pathfinding, the low-level data are loaded to achieve specific results. This step is referred to as "hierarchical pathfinding operations."



Region info



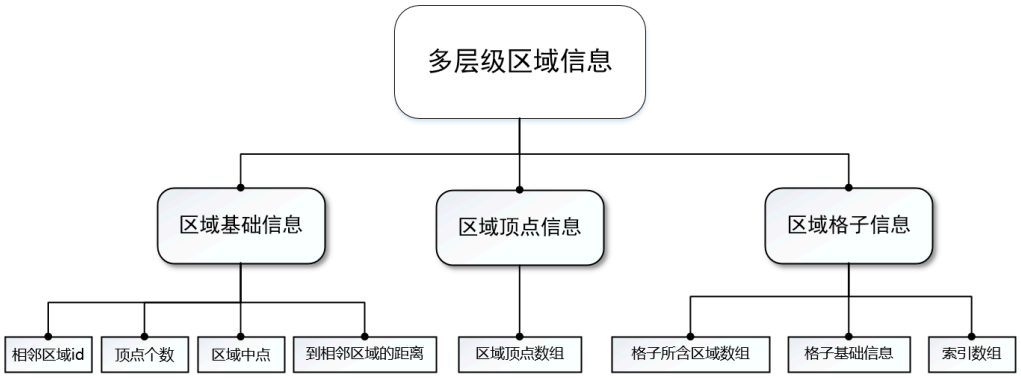
Poly info

3. Technical Implementation

3.1 Multi-level Data Construction

3.1.1 Area Data Construction

This step primarily involves constructing high-level area data based on Recast. The area information constructed for multi-level pathfinding includes "basic area information," "area vertex information," and "area grid information."



Hierarchical Area Information

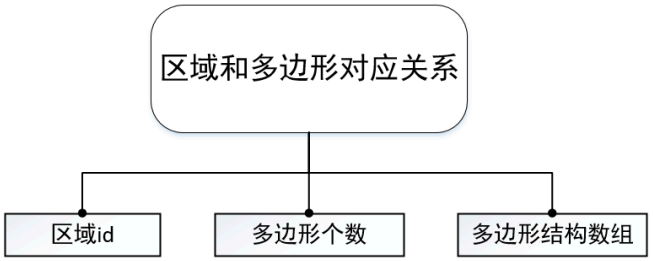
The foundation for constructing multi-level area data is based on the contour information generated by the watershed method used in Recast's pathfinding data. This includes basic area information and all edge contour information. The edge contour information records specific area contour vertex information and the relationships between connected areas. Using this information, most of the multi-level pathfinding area information can be constructed, although some details require individual construction, such as:

- Area Centroid: Calculate the centroid of the area shape, and use the location of the centroid as the area's central point.
- Distance to Adjacent Areas: By traversing the vertex data in the edge contour information, identify which two vertices belong to the same adjacent area, use them as the adjacent edge, and calculate the sum of the distances from the midpoint of this edge to the centroids of the two areas. This sum is recorded as the distance to the adjacent area.
- Area Grid Information: The scene is subdivided into grids of a certain granularity. Bounding boxes are used to determine the grid in which an area is located, which is used subsequently to ascertain the positions of target points.

3.1.2 Construction of Area and Polygon Relationship Data

This step primarily involves constructing the relationships between high-level area information and low-level polygon information.

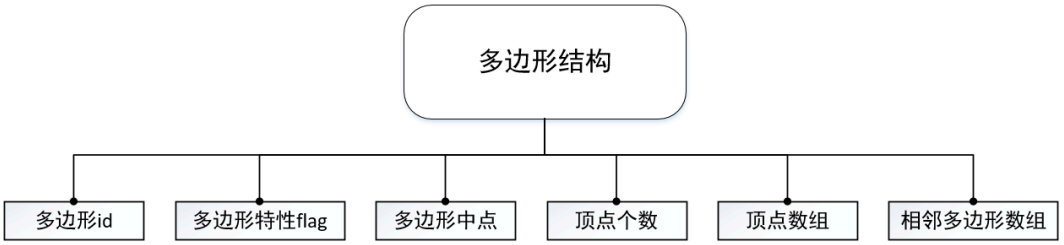
◦



Relation between area and polygons

3.1.3 Construction of Polygon Vertex Data

This step involves constructing low-level polygon information based on the foundation provided by Recast.



polygon structure

All information for the areas constructed in the multi-level pathfinding can be obtained from the polygon information generated in the Recast process. During the storage of polygon information, two specific data structures are constructed to facilitate subsequent access to polygon information:

Index Array: Organized by area, this records the size of the data stored for each area and creates an index that is placed at the beginning of the file for rapid access.

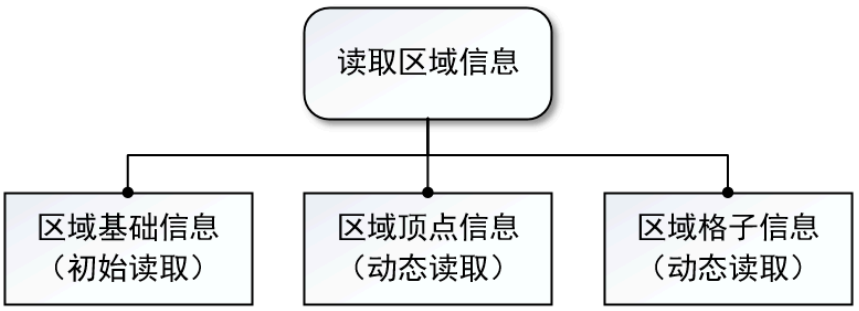
Special ID: Uses the Cantor pairing function to combine area IDs and polygon IDs, which aids in the quick retrieval of data in subsequent processes.

3.2 Multi-level Pathfinding Operations

3.2.1 Inter-Area Pathfinding

This step involves using high-level area information for pathfinding.

Region detour



Read polygon information

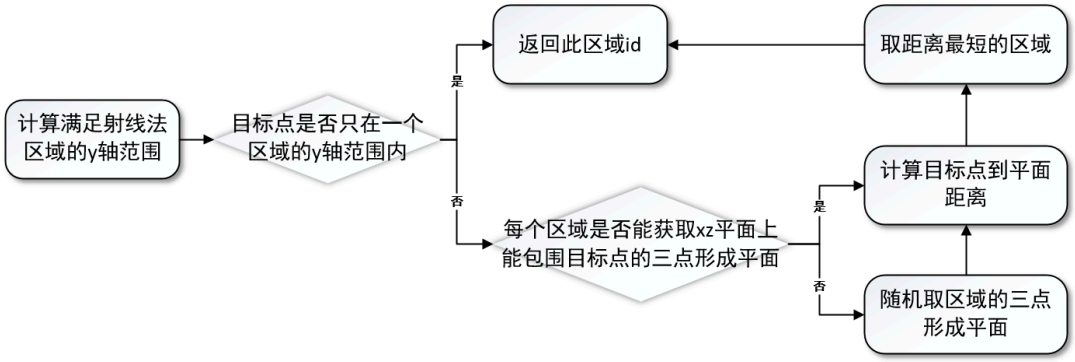
3.2.1.1 Reading Area Information:

The "basic area information" is primarily used for pathfinding within the region. "Area vertex information" and "area grid information" are necessary for determining the target point's location within the region.

3.2.1.2 Calculating the Target Point's Region:

The grid coordinate in which the target point resides is calculated based on the distance from the target point to the original point of the grid in the "area grid information." Due to the possibility of complex multi-layered 3D structures in the scene, which may include multiple planes, the spatial location of the target point may be close to several areas. To accurately determine the region to which the target point belongs, a ray-casting method is applied across all areas contained within the grid to assess whether the target point lies within the xz-plane of any region.

This approach also addresses situations where different areas exist at different y-heights. If multiple areas satisfy the ray-casting criteria, a detailed comparison of these areas is performed.



Detailed comparison

The detailed comparison is divided into two steps. First, it assesses whether the y-range of the area includes the target point, and if only one area includes it, that area is selected. Second, if no area includes the target point or if all areas include it, a plane is formed using three points around the target point, and the distance from the target point to this plane is calculated. For this, all vertices of the area are read, and based on the xz coordinates of the target point, which divide the area into four quadrants, one vertex from the contour of the area is taken from each quadrant, totaling four points. If there are three or more such points, any three of these points (or any three of the four) are used to form the plane. These three points are considered the three points surrounding the target point. If there are not three points meeting this criterion, three points are randomly selected to form the plane. Ultimately, the area whose plane is closest to the target point is selected as the target area.

3.2.1.3 Calculating the Area Path:

Pathfinding between areas is conducted using the centroids of the areas and information about adjacent areas. The A* algorithm is used where f = g + h, with h being the Euclidean distance between the centroids of two areas, and g being the inter-area distance previously calculated and stored in the basic area information. Here, the inter-area distance is the distance from the centroid of one area to the midpoint of the border shared with an adjacent area. The final output is the sequence of areas traversed from the start point to the endpoint, providing the preliminary results of the pathfinding.

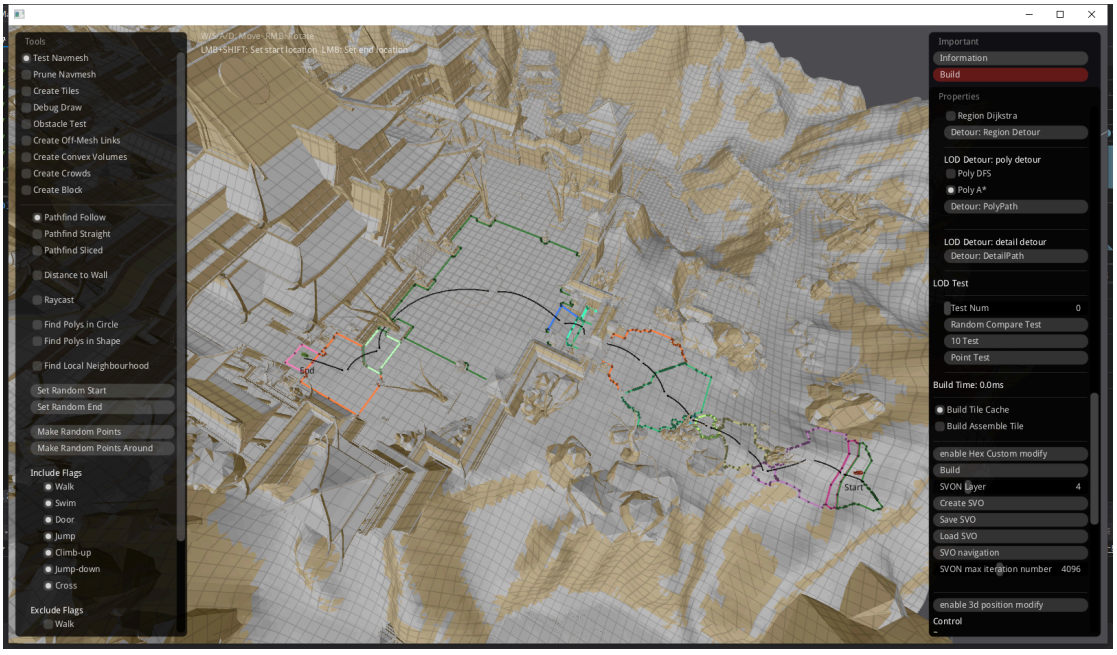
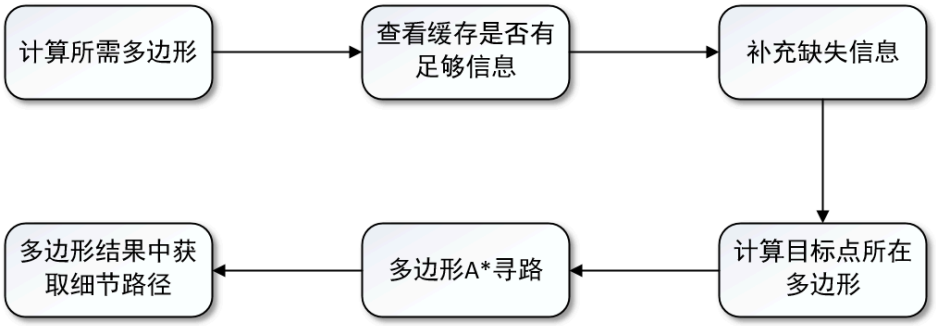


图 9. 区域寻路结果

3.2.2 Based on the results of the inter-area pathfinding, conduct specific pathfinding.



The following detour process

3.2.2.1 Calculating Required Polygons, Obtaining Related Polygon Information:

In previous steps, by using recorded index array data, polygon information for relevant areas is dynamically retrieved, and polygon IDs are recorded.

3.2.2.2 Checking for Required Polygon Information in Cache:

The Least Recently Used (LRU) caching mechanism is utilized to store polygon information in the cache. This step involves checking whether all the polygon information needed for the current pathfinding is present in the cache.

3.2.2.3 Retrieving Polygon Information Not Present in Cache:

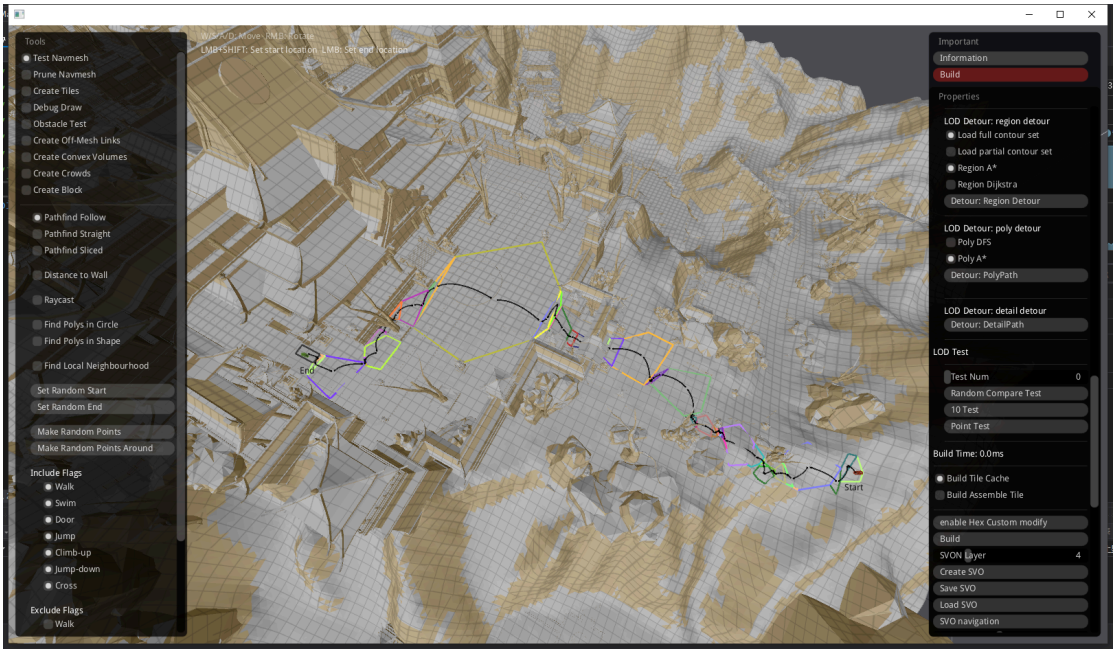
Following the previous step, polygon information not found in the cache is recorded. This caching mechanism stores polygon information from previously searched areas to avoid loading all data during map loading, thereby saving the required polygon information in the cache to speed up the pathfinding process.

3.2.2.4 Calculating the Polygons at the Start and End Points:

Since the starting and ending area IDs have already been obtained and all areas are considered to be on a single plane, a simple ray-casting method can be used to identify the starting and ending polygons. All polygons are assigned a unique ID to facilitate subsequent pathfinding.

3.2.2.5 Using the A Algorithm for Polygon Pathfinding:*

Pathfinding is performed based on the midpoints of polygons and their adjacency information. The A* algorithm uses the Euclidean distance to calculate heuristic values, offering high precision and accuracy. The results of the A* algorithm are recorded as the path traced through the polygons.



Polygon detour result

3.2.2.6 Using the Inflection Point Method to Find the Final Path Based on the Polygon Path:

Similar to the final steps in Recast pathfinding, the inflection point method is used within the polygons to determine the final route waypoints. The path points are then recorded and output. This method involves identifying key changes in direction or critical points along the path within the polygons to streamline and define the most efficient route.

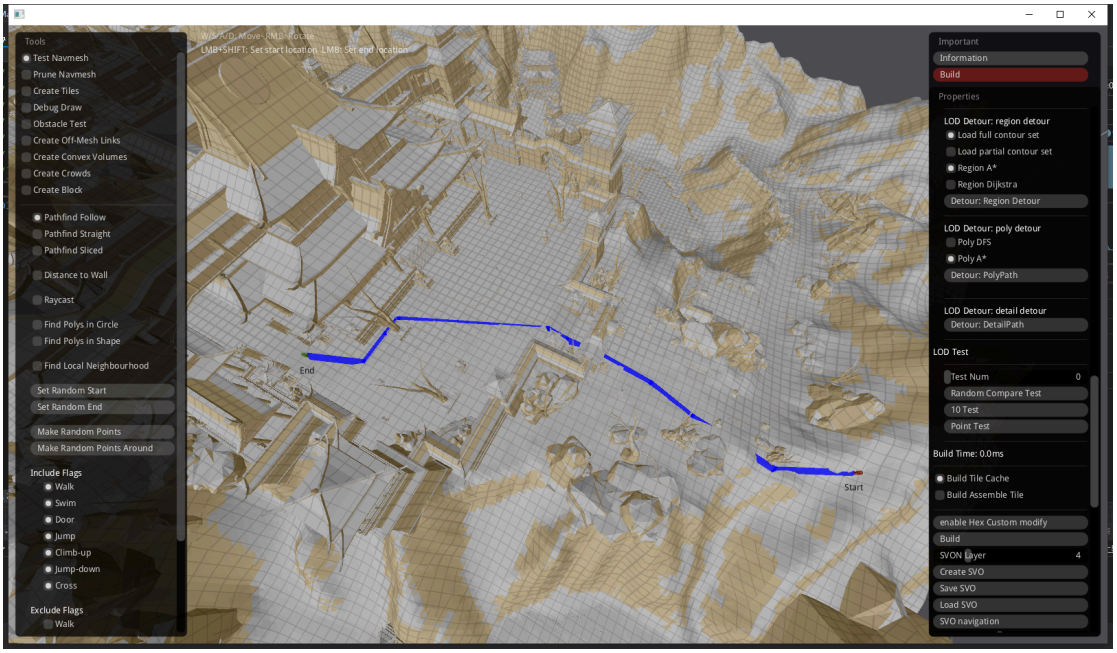


图 12. 细节路径寻路结果

3.3 Pathfinding Logic in Actual Game

3.3.1 Creating NavigateMap and NavigatorComponent During Scene Loading

When a game scene is loaded, the system initializes the NavigateMap and NavigatorComponent. These components are essential for managing pathfinding operations within the game environment.

3.3.2 After Creation, Check for Hierarchical Pathfinding Data

Once the components are created, the system checks whether the current scene contains hierarchical pathfinding

data. If such data are available, they are loaded into the cache. At the same time, any existing Recast cache data are unloaded, and the logic is switched to utilize hierarchical pathfinding.

3.3.3 Pathfinding Process for Players
When a player initiates pathfinding:

The target point's area within the scene is identified.
Area-level pathfinding is performed. If a valid path through the areas is found, indicating that the target point is reachable, the corresponding polygon data for those areas are loaded.
After the polygon data are fully loaded, polygon-level pathfinding takes place.
The final path points are generated using the inflection point method based on the polygon pathfinding results. For details on the process, see section 3.2.
3.3.4 Player Movement Logic During Pathfinding (Tick Logic)
During the pathfinding 'tick' (a cycle or iteration of game logic updates):

The system moves the player based on the results calculated in 3.3.3.
Using the polygon data from the pathfinding results, the player's current height is assessed. The current pathfinding polygon where the player is located is used to determine the player’s height on the pathfinding map.
Collision detection is then performed to ascertain the player's real-world height based on this pathfinding height.
3.3.5 Player Movement to Destination
Following the outlined process, the player moves to the designated endpoint. This final movement is governed by the pathfinding results and real-time adjustments during gameplay, ensuring the player navigates through the virtual world effectively and accurately.

This sequence of operations ensures that pathfinding in the game is both efficient and accurate, taking into account both area-level strategies for long-distance navigation and detailed polygon-level adjustments for precise movement and interaction within the game environment.

4. Application Result

Map	Recast Memory(KB)	Lod Memory after 3000 detours(KB)	ratio(%)	Diff between Lod and Recast%)
Saibei	68442	9739	14.23	2.73
Xiyu	60900	11004	18.07	1.23
Jiangnan	36700	4165	11.35	9.76
Jiangnan2	44878	1798	4.01	7.03
Guanzhong	39719	8705	21.92	9.36
Nanhai	35748	7083	19.81	0.46
Xueshan	3298	657	19.92	11.16

Map	Recast data size(MB)	Lod data size(MB)	ratio(%)	Recast detour time(ms)	Lod detour time(ms)	ratio(%)
Saibei	65.2	20.6	31.60	0.34	2.5	735.294
Xiyu	56.8	21.4	37.68	0.38	3.29	865.789
Jiangnan	32.7	13.5	41.28	0.35	1	285.714
Jiangnan2	41	10.4	25.37	0.39	1.86	476.923
Guanzhong	39.3	12.8	32.57	0.22	1.46	663.636
Nanhai	31.8	11.1	34.91	0.38	2.82	742.105
Xueshan	1.702	1.01	59.34	0.12	0.24	200.000

Tables 1 and 2 list the test results for the multi-level pathfinding algorithm (abbreviated as Lod). The tests were conducted using a cell size of 0.3 and cell height of 0.2 to generate Navmesh and multi-level data, respectively. The tests were performed 3000 times using a script, and the average values were recorded in the tables. From the key findings, we can see that the memory usage for multi-level pathfinding only requires up to 30% of that needed for the Recast project. Although it takes longer than Recast, the speed is within an acceptable range. The error rate comparison here is based on "global random points compared to Recast," but the pathfinding maps generated by multi-level pathfinding use a filtering algorithm, which differs from Recast. Tests conducted within the game are normal, and this error rate is provided for reference only.

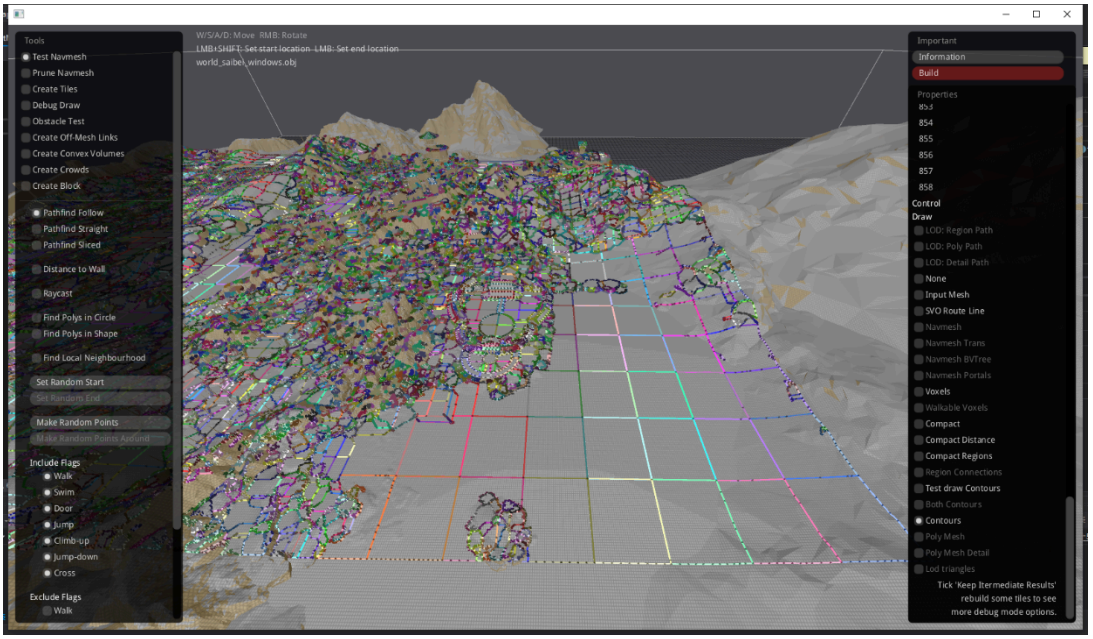
It should be noted that the Lod pathfinding data was generated after rewriting the process of generating Navmesh using the Messiah engine's RecastMapEditor. The modifications include:

After generating heightfields in segments, these segments are merged to create a complete heightfield. Based on this complete heightfield, open heightfields, areas, contours, and polygons are generated. This step significantly reduces the time required to generate pathfinding data in large maps using Solo mode, enabling the creation of contours and other data needed for multi-level pathfinding in large maps.

When generating areas and contours, the data structure of Recast is adjusted by replacing 'short' with 'int' and modifying some of the logic in data generation. This change aims to resolve issues with the original Solo mode, where there were limits on the number of areas and polygons that could be generated, preventing data generation for large maps. This ensures that large scenes can still generate Lod pathfinding data normally.

For excessively large areas, if they exceed a threshold, they are divided into four using their AABB bounding box, and the division steps are repeated, adding them as new areas to the overall data. This division is necessary because overly large areas do not meet the requirements for area pathfinding and would exceed the range in

subsequent processes. This division ensures that the final effects meet expectations.



Region split for large scale map

Restrictions are applied to the generation of the pathfinding map areas, incorporating an algorithm to filter out the lower parts of the map and prevent the generation of pathfinding maps for inaccessible rocks and the interiors of objects. This effectively limits the overall size of the pathfinding data.

5. Summary

This article introduces the multi-level pathfinding algorithm attempts made in the game "One Dream Jianghu," specifically focusing on the pathfinding module. The introduction of this algorithm covers two main aspects: data construction and pathfinding operations. In data construction, multi-level pathfinding cleverly builds on the existing information to prepare for subsequent steps. In pathfinding operations, various optimizations ensure accurate results while reducing memory usage and increasing speed. The proposed algorithm offers the following main advantages:

Reduced Memory Usage: Memory consumption for pathfinding data is reduced by 70%-90%, lowering the probability of crashes due to insufficient memory on less advanced devices.

High Pathfinding Precision: The algorithm can precisely locate target positions, producing optimal and shortest pathfinding results.

Increased Speed: Using caching and optimized algorithm logic keeps the pathfinding time within an acceptable range.

However, the project still has some shortcomings, which will be gradually addressed and improved in the future:

Further Optimization of Pathfinding Time: Continued efforts to reduce the time it takes to complete pathfinding.

Differences in Pathfinding Routes: When using Lod to generate area information, the region divisions differ from those created by the tiled Navmesh, potentially leading to different pathfinding routes compared to the original Recast pathfinding. Currently, there is no effective solution for minor deviations in paths, but these do not affect practical usability.

Data Compression: There is potential to compress the size of files needed for pathfinding, reducing the storage requirements.

These aspects highlight the progress and areas for development within the multi-level pathfinding approach used in "One Dream Jianghu."

Links

[1] https://km.netease.com/team/messiah/wiki/page/103111

[2] https://upcommons.upc.edu/bitstream/handle/2117/98738/Pelechano_HNAstar_prePrint.pdf
[3] https://km.netease.com/team/hex_sdudio/article/444857
[4] https://km.netease.com/team/hex_sdudio/article/465531
[5] https://km.netease.com/team/hex_sdudio/article/390249