Computers & Graphics xxx (xxxx) xxx

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

ELSEVIER

Technical Section

Multi-agent parallel hierarchical path finding in navigation meshes (MA-HNA*)☆

Q1 Vahid Rahmani, Nuria Pelechano*

Universitat Politcnica de Catalunya, Barcelona, 08034, Spain

A R T I C L E I N F O

Article history:
Received 28 March 2019
Revised 26 October 2019
Accepted 28 October 2019
Available online xxx

Keywords:
Multi-agent path finding
Hierarchical search
Parallel path finding

A B S T R A C T

One of the main challenges in video games is to compute paths as efficiently as possible for groups of agents. As both the size of the environments and the number of autonomous agents increase, it becomes harder to obtain results in real time under the constraints of memory and computing resources. Hierarchical approaches, such as HNA* (Hierarchical A* for Navigation Meshes) can compute paths more efficiently, although only for certain configurations of the hierarchy. For other configurations, the method suffers from a bottleneck in the step that connects the Start and Goal positions with the hierarchy. This bottleneck can drop performance drastically. In this paper we present two approaches to solve the HNA* bottleneck and thus obtain a performance boost for all hierarchical configurations. The first method relies on further memory storage, and the second one uses parallelism on the GPU. Our comparative evaluation shows that both approaches offer speed-ups as high as 9x faster than A*, and show no limitations based on hierarchical configuration. Finally we show how our CUDA based parallel implementation of HNA* for multi-agent path finding can now compute paths for over 500K agents simultaneously in real-time, with speed-ups above 15x faster than a parallel multi-agent implementation using A*.

1. Introduction

Path planning for multi-agents in large virtual environments is a central problem in the fields of robotics, video games, and crowd simulation. In the case of video games, the need for highly efficient techniques is crucial as modern games place high demands on CPU and memory usage.

Path finding should provide visually convincing paths for one or many autonomous agents in real time. Typically, it is not necessary to obtain the optimal path for all agents, instead use paths that look convincing to the viewer and can be computed within strict time constraints (to support 25 frames per second considering all other computations required in a game such as rendering, physics simulation, and AI).

The problem of path finding can be separated from local movement, so that path finding provides the sequence of cells to cross in the navigation mesh, and other methods can be used to set way-points and to handle collision avoidance against other moving agents in the cell [1].

In this paper, we focus on abstraction hierarchies applied to multi-agent path-finding to improve performance. A general notation consists of labelling the hierarchy as levels or layers in ascending order, with the lowest, L0, being the un-abstracted map in the game space, and consecutive layers numbered L1, L2 and so on representing higher levels of abstraction. The key idea consists of performing a search at a high-level, which is then "filled in" with more refined sections of the path at lower levels, until a complete path is specified.

Typically a high-level solution can be rapidly calculated, and the challenge lies in inserting the specific Start (S) and Goal (G) positions to connect them with the high-level graph. The literature in this field shows that the S/G (Start/Goal) connection step can become a bottleneck in both 2D grids [2] and Navigation Meshes [3].

There are many techniques that have shown performance improvements for the case of 2D regular meshes without a large memory footprint [4,5]. However, general navigation meshes consisting of convex polygons of different complexity present more challenges due to their irregular nature (i.e. not all the cells have the same size and edge length) [6]. In this work we propose two approaches to eliminate the existing bottleneck in hierarchical path finding for general navigation meshes, and evaluate their advantages and limitations in terms of both memory usage and performance improvements. The proposed solutions provide a large speed up for all configurations of the hierarchy, and makes

☆ This article was recommended for publication by Prof A. Jacobson.

* Corresponding author.
E-mail addresses: v.rahmani2015@gmail.com (V. Rahmani), npelechano@cs.upc.edu, npelechano@lsi.upc.edu (N. Pelechano).

https://doi.org/10.1016/j.cag.2019.10.006
0097-8493/© 2019 Published by Elsevier Ltd.

our new HNA* algorithms viable for even larger environments than before. Our solution can also be combined with multi-agent simulation, to handle several hundred thousand agents computing paths simultaneously in real time.

2. Problem formulation

A world map is typically given as a polygon soup. In order to have agents navigating a world map, it is necessary to find a representation of the walkable space. This can be done with a navigation mesh, which represents the walkable space as a collection of convex polygons called cells (could be triangles or polygons of more than three sides), where borders between adjacent cells are called portals [7]. Agents can move within any two points of a cell or cross portals to move between adjacent cells, without colliding with the static obstacle borders of a cell. This representation can be expressed as a graph $G=(N,E)$, where the collection of cells or convex polygons are the nodes or vertices of the graph $N=<p_0,p_1,\ldots,p_n>$, and the portals are the edges E, with each edge e_{ij}, corresponding to the edge between two adjacent polygons p_i and p_j. The cost of an edge $c(e_{ij})$is calculated as the distance between the center of polygon p_i to the center of polygon p_j, and thus it is always a positive value. Path-finding involves finding a path $P=\langle S,\ldots,u,\ldots,v,\ldots,G\rangle$ which is a sequence of nodes connected by edges, from the starting position S to the goal position G. The cost of a path $c(P)$ is the sum of all the costs assigned to the edges along the path P, and since all edges costs are positive values, the cost of a path will always be a positive value. The shortest path between S and G is the path of minimum cost among all possible paths. A* performs an informed graph search, by computing for each node being explored the function $f(x)=c(x)+h(x)$, where $c(x)$ is the current cost from S to node x, and $h(x)$ is the heuristic that estimates the optimal cost of the path from x to G [8]. When dealing with maps, $h(x)$, can be computed as the Euclidean distance between the position of the center of node x, and the position of the center of node G. With this heuristic, A* can always find the optimal path, which is the path of minimum distance.

Each level of the hierarchy Lx, $x>0$, is represented by a new graph G_x which is created by merging μ connected nodes from G_{x-1} (the value of μ is decided by the user). The new graph $G_x=(N_x,E_x)$, consists of a set of nodes $N_x=\langle n_x^0,n_x^1,\ldots,n_x^m\rangle$, where each node in G_x is a subgraph of μ connected nodes from G_{x-1}, so that $n_x^i=\langle n_{x-1}^j,n_{x-1}^k,\ldots,n_{x-1}^l\rangle$. Edges E_x in G_x are the subset of edges from G_{x-1} that connect two nodes n_x^s and n_x^d, where $s\neq d$.

Definition 2.1. An *Inter-edge*, ι_x^{sd}, in G_x is an edge e_{ij} from G_{x-1} that connects two nodes n_{x-1}^i and n_{x-1}^j, such that n_{x-1}^i is inside n_x^s, n_{x-1}^j is inside n_x^d, and $s\neq d$.

For those edges e_{ij} from G_{x-1} that connect two nodes n_{x-1}^i and n_{x-1}^j, such that both n_{x-1}^i and n_{x-1}^j are inside n_x^s, they become internal edges of node n_x^s. Therefore, there is no loss of connectivity between G_{x-1} and G_x, since all the set of edges in E_{x-1} are now either internal edges of nodes n_x^s in G_x or *inter-edges* in G_x.

These concepts are shown in Fig. 1. In the case of L1, the merged nodes from L0 are polygons of the navigation mesh. Fig. 2 shows an example of a simple navigation mesh from level L0 to L3. Colors are used to represent nodes at each level, so we can appreciate how each navigation mesh polygon turns into a node at L0, and then several connected polygons from L0 are merged in one larger node at L1, and similarly for L2.

The graph G_x contains a partition of G_{x-1}, with nodes at Lx being groups of adjacent nodes from $L(x-1)$, and edges E_x being a subset of the edges of E_{x-1}. Each node n_x can be traversed by finding an internal path between a pair of *inter-edges*. Such internal paths are represented by a sequence of polygons and can be precomputed and stored.

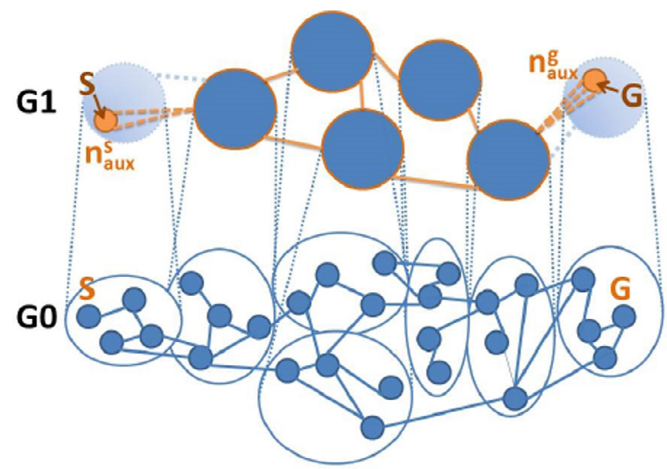

Fig. 1. Example of HNG with two levels and $\mu=4$. The orange circles and discontinuous links represent the temporal nodes and edges created after linking Start and Goal points to the HNG. This temporal graph is where the HNA* runs [3].

Definition 2.2. An *Intra-edge*, $\pi_x^{s(dk)}=\langle p_0,p_1,\ldots,p_k\rangle$, is a sequence of polygons from G_0 that represent the optimal path to traverse a node n_x^s between two *inter-edges* ι_x^{sd} and ι_x^{sk}. Therefore, $\pi_x^{s(dk)}=optimalPath(\iota_x^{sd},\iota_x^{sk})$. Its weight is computed as the sum of costs of the edges e_{ij} along the path, $c(\pi_x^{s(dk)})=c(e_{01})+c(e_{12})+\ldots+c(e_{(k-1)k})$, where e_{ij} is the edge between nodes p_i and p_j.

A node n_x^s will have an *intra-edge* for each pair of *inter-edges*. In order to find a high level path, we need a Hierarchical Navigation Graph, $HNG_x=(V'_x,E'_x)$, which captures the connectivity of G_x given by the relationships between *inter-edges* and *intra-edges*. In HNG_x, the vertices are all the *inter-edges* in the partition represented by G_x, $V'_x=\langle\iota_x^{sd},\iota_x^{dk},\ldots,\iota_x^{lm}\rangle$, and the edges, E'_x are *intra-edges*, $\pi_x^{d(sk)}$ connecting each pair of *inter-edges*, for which a path exists.

Note that HNG_x maintains the connectivity of the navigation mesh, but in a more compact representation, where only some edges are kept as nodes in HNG_x (those *inter-edges*, which depend on the hierarchical level L and the merging factor μ), and the shortest paths at $L0$ between those nodes are precomputed as *intra-edges*. Therefore HNG_x is built in a way that guarantees that the connectivity between polygons at L0 is kept regardless of the hierarchical configuration.

If a path, $P_0=\langle p_S,p_1,p_2,\ldots,p_G\rangle$, exists at G_0, then there will be a path at level Lx. Computing path finding in HNG_x gives as a result the path $P_x(S,G)=\langle\pi^S_{temp},\pi_x^{s(dk)},\pi_x^{k(sq)},\ldots,\pi_x^{r((m-1)m)}\pi^G_{temp}\rangle$. $P_x(S,\,G)$ is the high level path. The temporal paths, π^S_{temp} and π^G_{temp}, were created during the connect S and G steps, which computes a path at level L0 for the subgraph represented by the high level node S, and similarly for G. Therefore $\pi^S_{temp}=\langle p_s,p_0,p_1,\ldots,p_n\rangle$ where p_n is a polygon with one of the edges being the *inter-edge* that connects p_n with the first polygon in $\pi_x^{s(dk)}$. Extracting the sequence of polygons from each *intra-edge* $\pi_x^{i(jk)}$ we obtain the full sequence of polygons to traverse the navigation mesh between S and G (Proof in appendix A).

3. Related work

The most common approaches to speed-up path-finding, consist of either building some abstraction or hierarchy where path finding can be performed with smaller graphs (independently of the path-finding algorithm used), or else modifying the A* algorithm to gain

Please cite this article as: V. Rahmani and N. Pelechano, Multi-agent parallel hierarchical path finding in navigation meshes (MA-HNA*), Computers & Graphics, https://doi.org/10.1016/j.cag.2019.10.006