

#2 – Pathfinding is not ‘A-star’

White paper by Pierre Pontevia, CTO Kynogon

1. GOOD OLD PATHFINDING AND THE MAGIC A*

A game without Non Player Characters (NPCs) moving around is an interesting challenge for a designer! Moving is the foundation of action: an NPC that cannot move will not pick up objects, attack, hide, implement tactics, etc. Pathfinding, the technology required to move characters, is therefore a must for almost every game. This is probably the reason why game developers have been working on pathfinding for years and why it's extensively discussed in many articles and conferences. It is unusual to see a development team without pathfinding experience.

On the other hand, it is not unusual to see NPCs having troubles with their pathfinding. We often see characters blocked, lost, running against a wall, unable to go through a door, bumping into each others, falling into holes, etc.

The paradox probably comes from the fact that **pathfinding is not A***.

A* or equivalent algorithms are widely used throughout the game development community to solve the key question: how to go from one point of the map to another one? A* is an old and mature technology, extensively presented in most pathfinding articles and you can download code examples on the web: A* is not complex to implement. One might conclude that pathfinding is therefore easily solved.

A* is the trivial part of a pathfinder however - there is much more to pathfinding than A*: pathfinding is not path-planning. A lot of pending issues outside of path-planning remain:

- How to take into account constraints such as furtiveness when computing a path?
- How to follow a path computed by the path planner?
- How to take into account other NPCs when following a path?
- How to deal with dynamic evolutions of the world?
- How to ensure pathfinding overall performance?

This white paper's objective is to stress the challenges of 'traditional' pathfinding outside of A* and explains RenderWare A.I.'s ability to meet them. These challenges become critical as traditional workarounds will not suffice for next generation games. In order to remain focused, herd pathfinding (flocking), formation, etc. will not be considered.

2. TRADITIONAL TRICKS WILL NOT DO FOR NEXT GENERATION GAMES

In order to overcome some of the pathfinding complexity, developers have previously used tricks that impact game design as well as the production process:

- **Terrains are simplified.** Configurations that are not properly handled by the pathfinding algorithms are simply removed from terrain. If an artist has not integrated these constraints upfront, he will see developers coming and asking to remove objects, modify terrain, etc.
- **Scenarios are adapted:**
 - NPCs remain in limited areas and cannot move around the whole world. For example, if NPCs cannot properly use an elevator, they will not use it. There will be two groups, one upstairs and the other one downstairs.

- To avoid high pathfinding CPU consumption, developers reduce the number of active NPCs.
- Etc.
- **Pathfinding data are manually generated and tuned:** developer/artist will manually position points that will help NPCs properly go through a door, avoid an obstacle, ...
- Etc.

With next generation games, situation will not improve.

- Map sizes will significantly increase. Manual terrain data generation and tuning will become too time-consuming.
- Not only will maps be larger, also the number of NPCs will grow. Pathfinding performances will become an even more sensitive issue.
- Worlds will become more and more dynamic (with full physics), seriously challenging existing static pathfinding solutions.

3. TRAJECTORY COMPUTATION IN A STATIC WORLD

3.1 The magic A*: pathfinding is not path planning

A* and pathfinding are favorite topics of game development and have already been very well presented by many authors. A* (and equivalent) algorithms are efficient to explore a grid/graph and identify the shortest path to reach destination.

However pathfinding cannot be reduced to just trajectory computation. For RenderWare A.I., A* consumes less than 15% of CPU burnt by pathfinding and represents a very small portion of the code. Pathfinding is much more than path planning.

3.2 Pathfinding data requires an adequate world modeling

Trajectory computation relies on specific data. The most popular data models are grids (2D) or graphs of points and edges (3D). Generally, those models are manually generated. Somebody in the development team has to map all levels with points. When mapping, the trick is to have an even density, alongside an accurate description of the world. On one hand, the more points you put, the more memory and CPU pathfinding will consume. On the other hand, a low density description will not be sufficient for NPCs to properly move. This data must also take into account the fact that some zones are accessible only by jumping or crouching, etc.

As worlds are becoming larger and larger, manual data generation is going to become impossible. Next generation games will require an automatic pathfinding data generation process.

RenderWare A.I. proposes a specific modeling of the world. Game developers need either a world modeled from a rendering perspective (polygons) or from a physics perspective (collision meshes). They also need a model from a behavior perspective. RenderWare A.I. proposes such a modeling called the PathData. It is then used for pathfinding as well as advanced 3D perception (see white paper #1 Open the eyes of your Non Player Characters).

RenderWare A.I. offers a PathData Generator tool to automatically extract this modeling. PathData accurately describe the world and are optimized for minimum CPU and memory consumption. The generation process takes into account game engine specificities such as collision models and movement models. A manual edition of PathData is then possible for local customization.

3.3 Paths can be diverse: constrained pathfinding

Paths followed by characters may not always be the shortest ones. NPCs may need alternative paths, e.g.:

- they want to remain most hidden from enemies taking into account dark areas, fog, terrain topology, etc.
- they have to avoid dangerous zones
- they want to optimize energy consumption when they are tired

RenderWare A.I. proposes the concept of constraints. For example, furtiveness is a constraint. For each path a cost related to a constraint is computed. In the case of stealth paths, the more the path is exposed to enemies, the higher its cost will be.

Cost functions give NPCs the ability to choose between different kinds of paths.

4. TRAJECTORY COMPUTATION IN A DYNAMIC WORLD

If players and NPCs can move or destroy objects, they may block doors, accesses, etc. Pathfinding then needs to take into account these dynamic modifications of the environment, for example:

- if a bridge is destroyed, NPCs will find an alternative way to cross the river
- If a door is closed, they will look for a key or find another way
- If an elevator has not arrived, they will wait
- If objects block an entrance, they will go through the window

A pathfinding that handles dynamic objects is becoming more critical, as developers are using sophisticated physic engines to take control of dynamic objects. 'Real' physics also appears as a must for next generation games.

To handle dynamic modifications of the world, RenderWare A.I. proposes the PathObject concept. A PathObject is a dynamic object (elevator, door, teleporter, crane, etc.) that interferes with pathfinding. When moving, adding or destroying dynamic objects, an NPC or player changes the local topology properties and prevents or authorizes accesses. The path calculation integrates these modifications via PathObjects.

A PathObject is an "intelligent" object: it tells the NPC how to use it in order to properly progress. As an example, an elevator can be considered as a PathObject. When an NPC wants to use it, the PathObject will tell him which button to press in order to call the elevator, where to queue if other NPCs are already waiting, etc.

PathObjects are integrated during the automatic PathData generation (see 3.2 Pathfinding data requires an adequate world modeling). The modeling of the world will include dynamic objects so that path computations include PathObjects.

5. FOLLOWING TRAJECTORY

Trajectory is properly computed based on an adequate world modeling. It takes into account constraints and dynamic modifications. NPCs now need to properly follow the trajectory.

5.1 Smooth pathfinding

When following their path, NPCs movements have to be realistic. Trajectory needs to be smooth. If pathfinding is based on a graph of points, you do not want NPCs to pass through all the points of their path. Otherwise, they would look like robots sticking to predefined routes. Moreover, the way they follow their path will depend on their movement capabilities: a biped will turn much faster than a quadruped.

RenderWare A.I. proposes several modes for smoothing a trajectory. These different modes can be used to tune pathfinding consumption (see 6 - Optimization).

5.2 Dynamic avoidance

When several NPCs are moving around, for example in a crowd, they have to avoid each other. Potential collisions must be anticipated based on NPCs trajectories and bypass trajectories must be generated for each one. Bypass trajectories must take into account the presence of static obstacles. For example, in a corridor where two characters cannot walk at the same time, the bypass trajectory will not be a simple path around the character. The NPC may decide to wait until the other one is out of the corridor or compute a new path avoiding the corridor.

In RenderWare A.I., on one hand, a dynamic avoidance module is available to handle crowds. On the other hand, complex dynamic avoidance is tackled via the concept of PathObject (see 4 trajectory computation in a Dynamic world).

5.3 Path recovery

An NPC might be in a position where he cannot follow its path any more. For example:

- The NPC was pushed by an explosion or another NPC and he has fallen far away from his original path
- An obstacle has been moved and now blocks the NPC

The NPC first needs to identify when such an accident, for which a simple bypass trajectory will not help, has occurred. He then needs to re-compute a new path. Such a mechanism must of course be transparent: The NPC should not wait for ever to get a new path. It must however be efficient in order to avoid regular path computations that are time consuming.

RenderWare A.I. includes a path recovery system to manage accidents that may occur when following a trajectory.

5.4 Animation coordination

Following its trajectory, an NPC may choose between various animations. Imagine body guards protecting a VIP. They are following their path, strafing, walking backwards, etc. while aiming at several zones from where danger can come. NPCs can walk, run, jump, strafe and crouch, as their head and body rotation are independent. A proper pathfinding must be able to connect with a sophisticated animation system.

RenderWare A.I. can provide the animation system with any specific information it needs (destination, speed, head and torso orientation, etc.) Animations can then be properly selected depending on the NPCs situation and action, as well as animation system specificities.

6. OPTIMIZATION

Pathfinding remains one of the most consuming algorithms. With next generation games, pathfinding performances will become more critical. Large maps with lots of NPCs and dynamic objects demand high-performance pathfinding. Today solutions will need drastic improvements.

RenderWare A.I. proposes several optimization approaches that can be combined.

6.1 Time-slicing

Paths do not need to be computed every frame. Therefore, pathfinding computations can be spread over several frames. This time-slicing mechanism avoids CPU consumption peaks and is very efficient, in order to guarantee good performance.

When several time-consuming functions are called at the same time, a waiting list is created to ensure that all these functions will not be launched at the same time. Functions will be handled according to priorities that can be customized: the critical ones will be launched first. For example, pathfinding for visible NPCs can be considered as more critical than for not visible ones.

6.2 Pathfinding modes

RenderWare A.I. offers several modes for following a trajectory. For example, when NPCs are not visible to the player, you do not need a smooth trajectory with full dynamic avoidance. You can therefore apply a less time consuming mode that will still do the job.

6.3 Heuristic modes

Traditional A* heuristic is based on Euclidian distance, RenderWare A.I. proposes another heuristic that significantly accelerate A* computations.

6.4 Optimized data

Data has to be exhaustive enough to properly describe the world. A low density will not offer sufficient accuracy for NPCs to avoid holes, obstacles, etc. On the other hand, dense data will affect performances. An A* consumption is, for example, proportional to the square of the number of points of the graph.

The RenderWare A.I. automatic PathData generator has been designed to accurately describe a world as well as minimize data size.

6.5 Additional data

Memory and time consumptions are not independent. RenderWare A.I. can use optional pre-computed data to improve performances. If you allow pathfinding to use pre-computed data, you will get better run-time performances. On the other hand, if you do not use pre-computed data, RenderWare A.I. will require more CPU.

6.6 Streaming of pathfinding

Traditional optimization will not be enough for large maps. If developer map sizes and number of NPCs are a hundred times larger, even with drastically optimized algorithms he will never get an acceptable level of performance.

RenderWare A.I. will soon propose a hierarchical graph approach enabling pathdata streaming that is necessary for next generation games. This approach will divide PathData intelligently into several sectors. It will then be possible to load and unload pathfinding data according to developer specific mechanisms and priorities.

7. CONCLUSION

This white paper will not be the last one on the topic. Although pathfinding is a traditional topic of game development, it needs to be continuously pushed until it reaches the level of expertise that exists in rendering and physics. This is even more critical with next generation games, as they place serious demands on existing approaches. Issues that need to be addressed are much wider than only path planning. The magic A* does not answer all pathfinding complexity.