AUTODESK® KYNAPSE® MIDDLEWARE
WHITE PAPER

Pathfinding is Not A Star

White paper by Pierre Pontevia, senior director Product Development

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, or implement tactics. 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 is extensively discussed in many articles and conferences. It is unusual to find a development team without pathfinding experience.

On the other hand, it is not unusual to see NPCs still having trouble with their pathfinding. We often see characters that are blocked, lost, running into a wall, unable to go through a door, bumping into each other, or falling into holes.

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. Code examples are widely available on the web. A* is not complex to implement. One might conclude therefore that pathfinding is easily solved.

A* is the trivial part of a pathfinder however, there is much more to pathfinding than A*, and 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 make sure of pathfinding's overall performance?

The objective of this white paper is to stress the challenges of 'traditional' pathfinding outside of A* and to explain how Autodesk® Kynapse® middleware is able 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), and formation will not be considered.

2. Traditional tricks will not pass next-generation games

In order to overcome some of the pathfinding complexity, developers previously have 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 the terrain. If an artist has not integrated these constraints upfront, he will see that developers will ask that he remove objects, and modify the terrain.
- 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.

Autodesk®

  - To avoid high pathfinding Central Processing Unit (CPU) consumption, developers reduce the number of active NPCs.
- Pathfinding data is manually generated and tuned; the developer or artist will manually position points that will help NPCs, for example, properly go through a door or avoid an obstacle.

With next-generation games, the situation isn't improving.

- Map sizes significantly increase. Manual terrain data generation and tuning becomes too time-consuming.
- Not only maps are larger, but the number of NPCs grows as well. Pathfinding performances become an even more sensitive issue.
- Worlds 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 or graph and identify the shortest path to reach destination. However, pathfinding cannot be reduced to trajectory computation. For Kynapse, A* consumes less than 15% of CPU burned 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 and the most popular data models are grids (2D) or graphs of points and edges (3D). Generally, those models are manually generated by someone on the development team who has to map all levels with points. When mapping, the trick is to have an even density and an accurate description of the world. On one hand, the more points you have, 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. The data must also take into account the fact that some zones are accessible only by jumping or crouching, for example.

As worlds are becoming larger, manual data generation become impossible. Next generation games require an automatic pathfinding data generation process.

Kynapse offers 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. Kynapse includes such modeling called the PathData. It is used for pathfinding as well as advanced 3D perception (see white paper on perception, entitled Open the eyes of your Non Player Characters).

Kynapse offers the 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 specifics 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, such as when:

- they must remain hidden from enemies, taking into account dark areas, fog, terrain and topology
- they must avoid dangerous zones
- when tired, they want to optimize their energy consumption

Kynapse offers 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

4.1 Path Objects

If players and NPCs can move or destroy objects, they may block doors and entries. 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 a river
- if a door is closed, NPCs will look for a key or find another way
- if an elevator has not arrived, NPCs will wait
- if objects block an entrance, NPCs will go through the window

A pathfinding that handles dynamic objects is becoming more critical as developers are using sophisticated physics engines to control dynamic objects. ‘Real’ physics also appears as a must for next-generation games.

To handle dynamic modifications of the world, Kynapse uses the PathObject concept. A PathObject is a dynamic object (elevator, door, teleporter, or crane) that interferes with pathfinding. When moving, adding or destroying dynamic objects, an NPC or a player changes 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 it which button to press in order to call the elevator, and where to queue if other NPCs are already waiting. 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.

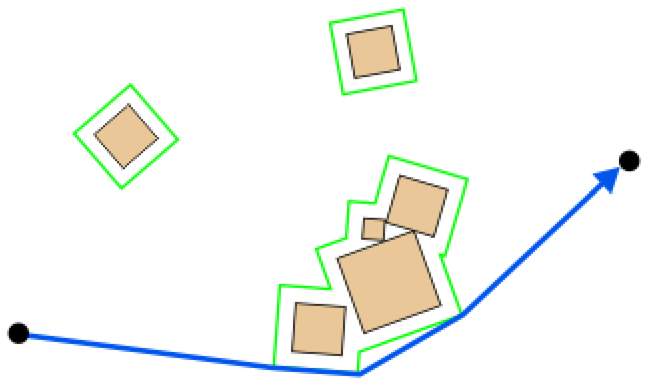
4.2 Dynamic pathfinding

With Kynapse 5.0, a fully dynamic pathfinding that can handle very dynamic terrains with thousands of movable objects was introduced.

The dynamic pathfinding helps Entities move around in dynamic, destructible worlds, where obstacles are constantly in motion. Dynamic pathfinding retrieves the outlines of dynamic obstacles and builds up complex collections of these obstacles. When an Entity is blocked by one object or aggregates of these objects, the dynamic pathfinding helps to construct and follow a path that goes around them.

Dynamic objects can create complex aggregates, and pathfinding can handle all combinations and configurations of obstacles.

Example of a trajectory around an aggregate of dynamic obstacles



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 paths, NPCs' movements must be realistic and trajectory must 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.

Kynapse includes 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 NPC is out of the corridor or compute a new path avoiding the corridor.

In Kynapse, a dynamic avoidance module is available to handle crowds. 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 it cannot follow its path anymore. For example:

- NPC was pushed by an explosion, or it and another NPC have fallen far away from their 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 work, has occurred. It then needs to re-compute a new path. Such a mechanism must of course be transparent; the NPC should not wait forever to get a new path. It must however be efficient in order to avoid regular path computations that are time-consuming. Kynapse 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 Very Important Person (VIP). They are following their path, strafing, and walking backwards while aiming at several zones from where danger can come. NPCs can walk, run, jump, strafe, and crouch, and their heads and body rotations are independent. A proper pathfinding must be able to connect with a sophisticated animation system.

Kynapse provides the animation system with any specific information it needs (destination, speed, head and torso orientation). 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 become more critical. Large maps with lots of NPCs and dynamic objects demand high-performance pathfinding. Kynapse offers 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 allow for good performance. When several time-consuming functions are called at the same time, a waiting list is created to make sure that all 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 invisible ones.

6.2 Pathfinding modes

Kynapse 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 Optimized data

Kynapse automatic PathData generator has been designed to as accurately as possible describe a world as well as minimize data size. Data has to be exhaustive enough to properly describe the world. A low density will not offer sufficient accuracy for NPCs to avoid holes and obstacles. 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.

In addition, Kynapse PathData includes a specific modeling of the terrain that offers a more reliable and CPU efficient result than traditional ray casting techniques. This modeling is called the AImesh and is a description of the borders of the world. It is automatically generated via the PathData Generator.

6.4 Streaming of pathfinding

Kynapse offers a multiple graph approach enabling PathData streaming. With this approach PathData are divided into several sectors. It is then possible to load and unload pathfinding data according to developer specific mechanisms and priorities.

6.5 Multithreading and multicore optimizations

Kynapse pathfinding takes advantage of console multithreading and multicore capabilities. Pathfinding computations can be off-loaded to specific threads, core, or Synergistic Processing Units (SPUs). CPU consumption on the main thread, core and Power Processing Unit (PPU) is therefore drastically reduced.

7. Conclusion

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 complexities.