18

Choosing a Search Space Representation

Nathan R. Sturtevant

18.1 Introduction

The choice of a path planning architecture for a game will help determine what features the game can support easily, and what features will require significant effort to implement. There are good articles describing different types of path planning architectures [Tozour 04], and there have been debates in different forums [Tozour 08, Champandard 10] about the correct choice for a path planning architecture. Each choice comes with its own set of benefits and drawbacks, the strength of which will depend on the type of game being developed and the time allotted to developing the architecture. The goal of this article is to summarize and extend some of the arguments made for different architectures.

It is important to know that, for most games, all feasible path planning architectures are abstractions of the space through which characters can walk in the game. This is because the physics that are used to simulate the world are not directly used as the path planning representation. So, in some sense, much of the debate here is related to what representation most closely matches the underlying physics of the game world.

This article focuses on the primary representations: grids, waypoint graphs, and navigation meshes. We assume that most readers are familiar with these representations, as they are probably the most common architectures used today. Furthermore, examples of several of these architectures can be found in this book. But, for reference, an example map is shown in Figure 18.1(a). Figure 18.1(b) shows the grid decomposition of the map, Figure 18.1(c) shows a waypoint graph on the map, and Figure 18.1(d) shows a triangle decomposition, which is a type of navigation mesh. This article is primarily directed

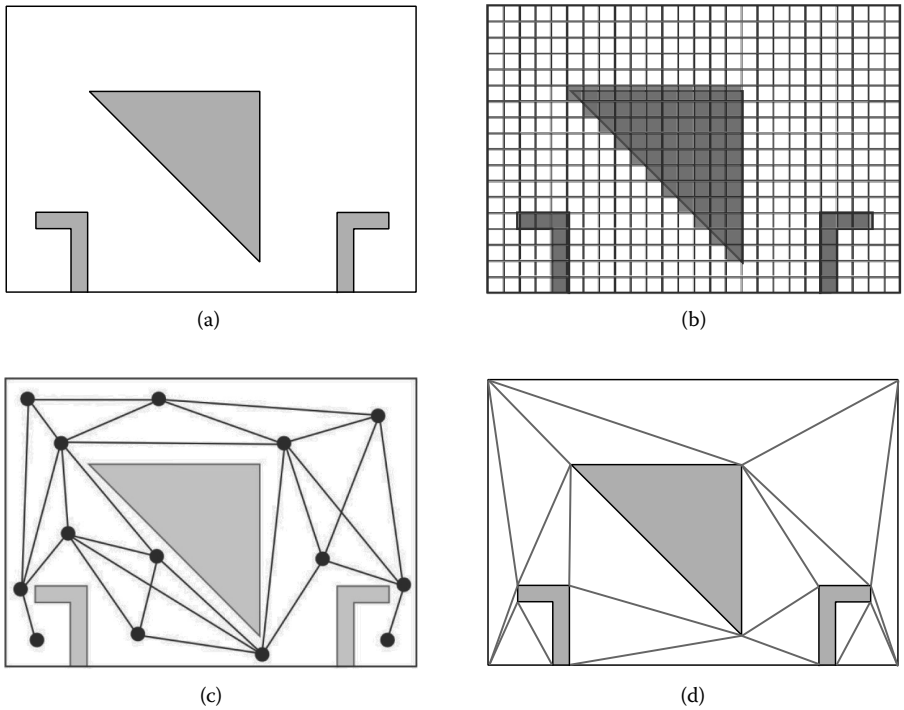

Figure 18.1

Three common world representations. (a) Original map, (b) grid decomposition, (c) waypoint graph, (d) nav mesh.

towards independent or small developers, as they are more likely to have a choice in the architecture they use.

18.2 Tasks

To begin, we briefly highlight the characteristics that we will consider when comparing path planning representations. These include memory usage, the ease of localization, planning, smoothing, path following, and dynamic modification of the representation. We also discuss the time required for implementation.

Memory usage is measured simply by the overhead of building and storing the representation of the map in memory. *Localization* is the process of moving from a spatial coordinate to a representation that is native to the path planning representation. When a user clicks the mouse, for instance, the coordinates of the click are recorded. This must then be converted into a grid cell or polygon in a navigation mesh. *Planning* is the cost of finding a valid path between two locations. *Smoothing* and *path following* is the process of taking a planned path and removing sharp turns or discontinuities to improve the overall quality. This can be done as part of planning, postplanning, or while the path is being followed by a character. *Dynamic modification* is the cost of performing changes to the representation on the fly, while the game is being played.

Note that the exact combination of each of these tasks depends on the game being created, and so the weight of each argument below depends on the importance of each task in your game. Grids, for instance, are a suitable representation for tower-defense games, but large open worlds in a MMORPG are usually too large for grids.

In addition to representing space for path planning, the representation can often be used for more generic queries to facilitate AI behavior. These may include positioning in battle, the best location for new buildings to be constructed, or the most protected location during battle. These queries are game-dependent, and so we will not directly consider them here.

18.3 Grids

The simplest implementation of a grid represents the world via an array of blocked and unblocked cells. More sophisticated implementations can include information on slope, terrain type, or other meta-information which is useful for planning. Grids traditionally represent only two-dimensional worlds, but can be used to represent three-dimensional worlds as well [Sturtevant 11].

The pros are:

- Grids are one of the simplest possible representations and are easy to implement. A working implementation can be completed in a few hours.
- A grid representation can be easily edited externally with a text editor. This can save significant tool-building efforts [Van Dongen 10].
- Terrain costs in grids are easy to dynamically update. For example, player-detected traps in Dragon Age: Origins are easily marked with a few bits in the relevant grid cells. It is easy for A* to account for these costs when planning, although the cost of planning will be increased if too many cells are re-weighted.
- Passable cells can be quickly modified in a grid in a similar way to terrain costs being updated.
- Localization in a grid is easy, simply requiring the coordinates to be divided by the grid resolution to return the localized grid cell.

The cons are:

- Grids are memory-intensive in large worlds. Note that a sparse representation can be used when the world is large, but the walkable space is relatively small [Sturtevant 11].
- Path smoothing usually must be performed to remove the characteristic 45° and 90° angles that are found in grid-based movement, although any-angle planning approaches can also be used [Nash et al. 07].
- Path planning in grids can be expensive due to the fine-grain representation of the world. This can be addressed using some form of abstraction [Rabin 00, Sturtevant 07].
- Grid worlds often contain many symmetric paths, which can increase the cost of path planning. Some techniques can be used to avoid this (e.g., [Harabor and Grastien 11]), but this can also be avoided with different state representations.

18.4 Waypoint Graphs

Waypoint graphs represent the world as an abstract graph. Importantly, waypoint graphs do not have an explicit mapping between nodes in the graph and walkable space. Waypoint graphs were widely used before the popularity of navigation meshes grew. While they have been criticized for their shortcomings [Tozour 08], they have also been praised for their strengths [Champandard 10].

The pros are:

- Waypoint graphs are relatively easy to implement.
- Waypoint graphs are easy to modify if the changes are known ahead of time. For instance, if a door in the world closes and is locked, it is easy for the developer to mark the edges in the graph that cross the opening of the door and block them when the door is shut.
- Waypoint graphs represent only a small fraction of the points found in a grid. This sparse representation of walkable space is both cheap to store and leads to inexpensive path planning requests.

The cons are:

- Path quality can suffer if there are not enough walkable edges in the graph, but too many walkable edges will impact storage and planning complexity.
- Waypoint graphs may require manual placement of nodes to get good path quality.
- Localization on waypoint graphs requires mapping between game space and the graph. If a character is knocked off of the graph, it may be unclear where the character should actually be within the waypoint graph.
- Because there is no explicit representation of the underlying state space, smoothing off the waypoint graph can result in characters getting stuck on physics or other objects.
- Dynamic changes are difficult when they aren't known ahead of time. If a character can create an unexpected hole in a wall, new connections on the waypoint graph are needed. However, it can be expensive to check all nearby connections to verify if they have become passable due to the changes in the map.

18.5 Navigation Meshes

Navigation meshes represent the world using convex polygons [Tozour 04]. A special case of navigation meshes are constrained Delaunay triangulations [Chen 09], for which the world is only represented by triangles. Note that grids can also be seen as a special case of navigation meshes, as both representations use convex polygons, but their usage is significantly different in practice.

The pros are:

- Polygons can represent worlds more accurately than grids, as they can represent non-grid-aligned worlds.

- With the accurate representation of a polygon it is easier to correctly perform smoothing both before and during movement. This accuracy can also be used for tighter animation constraints.
- Path planning on navigation meshes is usually fast, as the representation of the world is fairly coarse. But, this does not impact path quality, as characters are free to walk at any angle.
- Navigation meshes are not as memory-intensive as grids as they can represent large spaces with just a few polygons.

The cons are:

- The time required to implement a navigation mesh is significant, although good open-source implementations are available [Mononen 11].
- Navigation meshes often require geometric algorithms, which may fail in special cases such as parallel lines, meaning that implementation is much more difficult [Chen 09].
- Changes to navigation meshes can be difficult or expensive to implement, especially when contrasted with changes to grid worlds.
- Localization on navigation meshes can be expensive if poorly implemented. Good implementations will use additional data structures like grids to speed up the process [Demyen 06].

18.6 Conclusion

To conclude, each path planning architecture has its own strengths and weaknesses. The choice of an architecture should depend on the type of game being developed, the tools already available, and the time available for implementation and debugging. Many game engines ship with their own path planning representation, but for the cases where a new implementation must be performed, we summarize the pros and cons as follows:

Grids are most useful when the terrain is fundamentally 2D, when implementation time is limited, when the world is dynamic, and when sufficient memory is available. They are not well suited for very large open-world games, or for games where the exact bounds of walkable spaces are required for high-quality animation.

Waypoint graphs are most useful when implementation time is limited, when fast path planning is needed, and when an accurate representation of the world is not necessary.

Navigation meshes are best when there is adequate time for testing and implementation. They are the most flexible of the possible implementations when implemented well, but can be overkill for smaller projects.

Ultimately, the best representation is the one that minimizes developer effort and helps make the game-playing experience as compelling as possible. This may be different in any game, but being aware of the trade-offs between each architecture will help you make the best decisions on any new project.

References

[Champandard 10] A. Champandard. "Are Waypoint Graphs Outnumbered? Not in AlienSwarm." http://aigamedev.com/open/review/alienswarm-node-graph/, 2010.

[Chen 09] K. Chen. "Robust Dynamic Constrained Delaunay Triangulation for Pathfinding." Master Thesis, 2009.

[Demyen 06] D. Demyen. "Efficient Triangulation-Based Pathfinding." Master Thesis, 2006. Available online (https://skatgame.net/mburo/ps/tra.pdf).

[Harabor and Grastien 11] D. Harabor and A. Grastein. Online graph pruning for pathfinding on grid maps, *Proceedings of the AAAI Conference on Artificial Intelligence (2011)*, pp. 1114–1119. Available online (http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3761).

[Mononen 11] Mikko Mononen. "Recast." http://code.google.com/p/recastnavigation/, 2011.

[Nash et al. 07] A. Nash, K. Daniel, S. Koenig, and A. Felner "Theta*: Any-angle path planning on grids." *Proceedings of the AAAI Conference on Artificial Intelligence (2007)*, pp. 1177–1183. Available online (http://idm-lab.org/bib/abstracts/papers/aaai07a.pdf).

[Rabin 00] S. Rabin. "A* Speed Optimizations." In *Game Programming Gems*, edited by Mark DeLoura, Hingham, MA: Charles River Media, 2000, pp. 272–287.

[Sturtevant 07] N. R. Sturtevant, "Memory-efficient abstractions for pathfinding." *Artificial Intelligence and Interactive Digital Entertainment*, pp. 31–36, 2007. Available online (http://web.cs.du.edu/~sturtevant/papers/mmabstraction.pdf).

[Sturtevant 11] N. R. Sturtevant. "A sparse grid representation for dynamic three-dimensional worlds." *Artificial Intelligence and Interactive Digital Entertainment, 2011*. Available online (http://web.cs.du.edu/~sturtevant/papers/3dgrids.pdf).

[Tozour 04] P. Tozour. "Search space representations." In *AI Game Programming Wisdom 2*, edited by Steve Rabin. Hingham, MA: Charles River Media, 2004, pp. 85–102.

[Tozour 08] P. Tozour. "Fixing Pathfinding Once and For All." http://www.ai-blog.net/archives/000152.html, 2008.

[Van Dongen 10] J. Van Dongen. "Designing Levels without Tools." http://joostdevblog.blogspot.com/2010/12/designing-levels-without-tools.html, 2010.