Dynamic Path Planning in Minecraft

by

Christen Ford

A Thesis Proposal

submitted in partial fulfillment  
of the requirements for the degree of  
Bachelors of Science in Computer Science   
Baldwin Wallace University

2017

# Abstract

Path planning is one of the original fields in artificial intelligence. Abstractly, one can think of it as a simple navigational tool. However, its applications are far more widespread than that. It is a foundational block of artificial intelligence that more complex problems build upon. Three-dimensional path planning is a well-defined problem in computer science. The process is more complicated than when working in two-dimensions. Several factors to consider involve state space size, time complexity, and available hardware. Proposed is a method of computing shortest paths in three-dimensional space utilizing rapidly exploring random trees (RT-RRT\*) and advanced graphical hardware. We will further utilize research from several reputable sources in the field that pertain to topics such as voronoi diagrams, spatial subdivision graphs, and collision detection. This work carries importance to all fields of computing including robotics, operations planning, and other types of specialized software that utilizes advanced planning techniques. This work is also relevant to the development of autonomous vehicles and medicinal nano-drones. We will start with a review of the literature. Then we will proceed to our methodology and findings. Finally, we will present a summary of the research performed as well as final conclusions.

# Chapter 1: introduction

Path planning is a branch of artificial intelligence that is concerned determining routes for an artificial agent in both simulated and real environments. There have been many goals of path planning as well as many application domains. The most well-known involve a subset called shortest path algorithms. This type of path planning, as the name implies, is all about determining the shortest path from a source to a destination. Optimally, the best solution to this problem is to just follow a straight vector from the source to the destination. However, in practice, this is never achievable due to physical constraints such as obstacles and terrain or limits placed on the agent itself.

Another factor to consider when path planning is the number of dimensions to plan in. The obvious case and most easily approached is path planning in two dimensions. That is, path planning performed in a system that utilizes an (x, z) coordinate system. Many problems presented in two dimensions are easily solved by the various algorithms devised to do so, notably Dijkstra class algorithms. However, path planning becomes immeasurably more complex in terms of computation when the same problems are elevated to three dimensions. Unfortunately, this means that without advanced hardware, it becomes impractical to compute paths of any kind in three dimensions. This does not mean it cannot be achieved. Indeed, in constrained, regulated environments where inputs are limited, three dimensional paths can be determined. However, to have applicability to the real world, we must be able to at least model a spatially restricted sample of the world for pathing.

The final restriction on pathing in three dimensions is created by the enormous size of the sample space. With two dimensions of space, the sample space can be analyzed and a path can be computed in real time, regardless of complexity. If the sample space can be solved, a path can be determined. There currently exists only a small handful of path planning algorithms for computing paths in real time that are suitable for use in three dimensions. The most recent of these algorithms expands on the concept introduced by rapidly exploring random trees. As explained by Naderi et al. it extends the algorithm thorough constant resampling and rewiring to enable real time path planning (2015). This algorithm is known as rapidly exploring random trees or RT-RRT\*.

Given the recent advancements in power consumption for both central and graphical processing units. It is now possible to create a pathing algorithm that leverages advanced hardware to be able to create paths in three dimensions on the fly. While mobile computing has been a standby for decades, it has only recently reached a point where it matches the performance of traditional computing. This is further augmented by advancements in artificial intelligence as seen in platforms such as autonomous vehicles and the internet of things as well as robotics as showcased by Boston Dynamics.

Theoretically the improvements gained through distribution can be many. Most notable are improvements in the speed of calculations and the memory required to do so. Important is the fact that many modern graphical units involve computations across thousands of individual cores. Unlike a central processing unit which is better suited for throughput as opposed to speed, a graphics processing unit is capable by design of calculating at rates unachievable by a traditional processor. In addition, while central processing units contain multiple layers of localize memory cache, graphics processing units contain onboard with space thousands of times greater than the cache found in a traditional processor. Therefore, computation via the graphics pipeline weighs much more heavily on the development of artificial intelligence then does central processing. Cost is another factor to consider in this approach, however the cos of modern graphics hardware is now roughly equivalent to top tier computational units and is widely accessible to the public.

What this all leads to is the need for a new class of algorithms for a new world. Future algorithms will always build on those introduced in the past. This thesis serves to lay the framework for one such algorithm. This algorithm shall extend real-time rapidly exploring random trees to plan paths in real time, three-dimensional environments. It shall utilize advanced graphical hardware and processing techniques introduced in previous works. Finally, this algorithm shall be easily portable. Unlike several algorithms that utilize global data structures to compute paths, this algorithm shall be parallelized. This will allow us to compute paths across graphical cores, and possibly in the future across graphical units.

The goal of this thesis is to create an algorithm that under a set of specified criterion (specified in chapter three), can compute three dimensional paths in real time. However, even if the algorithm fails to meet these criterion, it will still classify as a success. Much of the work done in three-dimensional path planning is experimental. It is not as thoroughly researched as two-dimensional path planning. This leads to many failures in this field that further result in advancements at later times. As stated earlier, the need for three-dimensional path planning is rapidly increasing. A generalized approach to the planning problem can be utilized to solve problems in realms outside of computer science.

# Chapter 2: Literature Review

1. **LITERATURE REVIEW**

Creating three-dimensional paths directly in a three-dimensional plane is both temporally and spatially expensive. The direct approach to doing so is best left to high-powered machines as more widespread, cheaper computers cannot easily handle such a task. To solve this problem, we reintroduce the concept of spatial subdivision. This process takes a three-dimensional plane and splits it into a series of connected, two-dimensional planes using both voronoi diagrams and spatial subdivision. However, there is an additional drawback to this method. That is, computing voronoi diagrams is known to be exponential in run-time at the worst case and polynomial in run-time at the best case. To solve this, we leverage the fact that many household computers now contain discrete graphics processing units. They are built specifically to handle mathematically intensive work, that the central processing unit would otherwise struggle through. That said, Hoff et al. also present ways to compute voronoi diagrams utilizing GPU shaders. Camporesi and Kallman take the findings presented by Hoff et al. and apply them to computing shortest path maps. Finally, there is the issue of pathfinding in these newly created two-dimensional domains. The works presented by Ramires and Leonel, Leonel et al., Mitchell and Sharir, Naderi et al., and Burch and Weiskopf attempt to solve this.

As presented by Hoff et al., a voronoi diagram is simply a two-dimensional representation of a space that is split into voronoi regions utilizing carefully selected voronoi sites. The resulting boundaries, known in the literature as voronoi boundaries are, are then used by pathfinding agents. A common application of this technique is outlined in Hoff et al. to solve the piano movers problem, “The underlying idea is to treat the obstacles as sites. The voronoi boundaries then provide paths of maximal clearance between the obstacles” [1999]. As mentioned earlier, the issue with this technique is time. All algorithms that compute voronoi diagrams do so iteratively, the longer the time they can run for, the more accurate the representation of the space the resulting voronoi diagram will represent. A well-known algorithm for computing voronoi diagrams is known as Llloyds’ algorithm or more commonly, K-Means clustering. Hoff et al. utilize a parallelized version of this algorithm to compute voronoi diagrams using GPU shaders.

The techniques presented in Hoff et al. are then referenced in both Camporesi and Kallman [2014] and Mitchell and Sharir [2004] to further refine solutions to the three-dimensional pathing problem. Camporesi and Kallman build on the work done by Hoff et al. and present methods for computing shortest paths using GPU shaders. As stated in Camporesi and Kallman “Our method first relies on standard CPU algorithms for computing the shortest path tree of the obstacle set, and then applies the proposed shaders to encode the SPM in the frame buffer with arbitrary resolution” [2014]. This method has three stages. First, the environment space is preprocessed into discrete two-dimensional regions. Second, visibility graphs and shortest path trees are computed for each region. Finally, the shortest path map for each region is computed and the resulting paths for each region are adjoined together to create the overall shortest path map [2014]. Mitchell and Sharir apply voronoi diagrams to compute paths amongst stacked sets of axis-aligned polygonal shapes. They approach the problem by using spatial subdivision to represent the polygons as terrain. They then compute the shortest path using a topographical, top-down approach. This technique forces the agent to stick to flatter terrain, but could have the adverse effect of causing the agent to “sweep the terrain upwards” [2014] wherein the agent will generate a non-shortest path. As an aside to their work with terrain, Mitchell and Sharir also provide methods for computing shortest path distances over walls. They represent the wall as a series of interconnected lines in 3 space and then compute the shortest polygonal paths between them. The resulting sub paths create a path that appears to bend around the wall and is proven to be y-monotone.

Now that we have discussed methods for representing three-dimensional maps in two-dimensional space, we will discuss techniques for creating path from that space. Ramires and Leonel propose utilizing collision detection introduced in [2006] and later refined by Leonel et al. in [2008] to navigate three-dimensional space. They utilize a height based approach that automatically extracts needed information from the three-dimensional world and creates a minimum two-dimensional representation. “This is achieved by slicing the world with horizontal planes. For each slice, the height at which the slice was taken, as well as the height map obtained at that slice is kept” [2006]. This method works well in constrained, highly populated environments. When combined with the proper pathfinding algorithm, it can find paths quickly and efficiently. In their paper from 2008 Leonel et al. build on the information presented in their earlier paper and apply it to dynamic environments. The initial step in their process involves slicing to create spatial subdivision maps. After this step, they then compose the resulting planes together at connection points to create a single hierarchical navigation graph. This graph can then be utilized by A\*, in static environments, or in our case RT-RRT\* for dynamic environments. Even in a static environment RT-RRT\* is guaranteed to outperform A\* temporally, but will it will consume more space than A\*.

To Give some background into what a rapidly exploring random tree is, we present and review the findings in the work performed by Burch and Weiskopf They present algorithms both for computing rapidly exploring random trees and algorithms for visualizing them. Per Burch and Weiskopf an RRT as it is known in the literature is simply a tree that “… is computed incrementally by adding a new sample to the tree randomly, computing the least distant already existing sample in the tree by a distance function, and finally connecting both samples by as straight line that produces a new branch of the tree” [2013]. This process is repeated until the final path is computed. And since RRT is a probabilistic search method, it is faster than all the classical Dijkstra search methods and is also capable of parallelization. The visualization algorithms presented by Burch and Weiskopf allow for rendering an RRT on the screen as a graphical heat map, with earlier parts of the tree appearing on the screen more intensely than recently explored regions. The algorithm produces jagged edges around obstacles, but we are not concerned with the smoothness of the path, only its optimality.

Naderi et al. propose a modified version of Rapidly Exploring Random Trees call RT-RRT\* that can explore dynamic environments with adversaries. It works via incremental resampling and is explained best by Naderi et al. “…At each iteration, we expand and rewire the tree for a limited user-defined time. Then we plan a path from the current tree root for a limited used-defined amount of steps further” [2015]. While RT-RRT\* works great for dynamic environments, it has some drawbacks. First, is the spatial complexity of the algorithm. It stores the entire tree in memory and keeps it there until the path is found. For smaller maps, this is not an issue, but as the region RT-RRT\* is set to explore grows, the spatial requirements of this algorithm will grow with it, a drawback that is offset by the VRAM available on the GPU. Second, this method is optimized for bounded environments. An analysis of the map must be conducted beforehand to optimize RT-RRT\*. This is because it uses an ellipsoid method for resampling and rewiring. If the distances are too large, then this method will suffer from in time complexity.

1. **CONCLUSIONS**

As discussed in Ramires and Leonel [2006] collision detection can be made to be efficient simply by utilizing a divide and conquer approach of slicing and pathing. Camporesi and Kallman [2014] provided a method of quickly generating shortest path maps using GPU shaders through a three-step process. They also provided some potential drawbacks to consider, and how to avoid them. Mitchell and Sharir [2004] provide some techniques for generating shortest path map on polygonal terrain by simply pathing above it. They also introduce an approach to pathing over obstacles, useful in certain game genres.

Hoff et al [1999] show how voronoi diagrams can quickly be generated for both 2D and 3D worlds using graphics hardware. They do so by splitting the world at voronoi sites (represented as obstacles) and using the boundaries created between them to effectively path. Naderi et al. [2015] debut an advanced implementation of Rapidly Exploring Random Trees that allows paths to be computed in real time, dynamic environments. The algorithm they provide will constitute the pathing algorithm used in this thesis.

Leonel et al. [2008] showcased a method that demonstrated ow a 3D environment can be sliced into 2D chunks, that can then be connected through a single hierarchical navigation graph. This would enable easy traversal of the world by RT-RRT\*. Burch and Weiskopf [2013] show the beauty in rapidly exploring random trees. More importantly, they present novel techniques that will be utilized to help visualize the paths produced by RT-RRT\* in the 3D world.

# Chapter 3: Research Question and Methodology

1. **RESEARCH QUESTION**

Path planning in a multi-level, three-dimensional, dynamic environment is not easily solved. This has led to many different solutions to this problem that were created for specific application domains within this problem. Many solutions are not cross compatible. A solution for a specific application of this problem may not work for another, and in fact, this is often the case. Is it possible to adapt real-time rapidly exploring trees to better suit the needs of dynamic path planning in three dimensions? The algorithm should be able to compute paths on the fly Given this knowledge, we propose a method for solving the multi-level, three-dimensional, dynamic path-planning problem that will be generalized to apply to various application domains.

1. **METHODOLOGY**

This research will utilize the Project Malmo framework for the test environment for this research. Initially established by Microsoft Research on June 1st, 2015. Project Malmo is a framework that utilizes the Minecraft game environment for creating artificial agents. It was chosen as Minecraft is a discretized, constrained environment, where in-game agents have predictable behaviors. In addition, Minecraft contains both a world generator and the ability for players to create their own worlds. As the focus of this research will be on developing the path-planning agent and not the world it will navigate, it makes sense to utilize a prebuilt framework.

The agent must be able to navigate a set of small to medium sized predefined worlds with the goal of navigating to the goal destination. Before it may be utilized on randomly generated worlds, the agent must meet the following criteria for all testing worlds:

1. The agent shall be able to navigate diverse, multi-leveled terrain, some of which is hazardous to the agent such as lava.
2. The agent shall be able to navigate around opposing agents, avoiding them when necessary.
3. The agent shall not exceed greater than 25% deviance from the optimal path to the goal. As each world contains a predefined layout and goal, the only contributing factor to deviance from the optimal path shall be opposing agents and hazardous terrain.

To further strengthen statistical analysis of the path planning algorithm, it shall be run across each map a total of 1,000 times. This should enable us to filter out any bias introduced into the population via a small sample size. Because the algorithm will be run on the provided NVIDIA GPU cores, time constraints on the actual testing process are not a concern.

As Minecraft is a complex, three-dimensional multi-level environment, preprocessing must be performed before the agent may path the world. This thesis shall utilize a recently released real time variant of rapidly exploring random trees (RT-RRT\*). RT-RRT\* creates an elliptical sample space to create the tree from. To do this in the Minecraft environment, we will leverage the distinct unit boundaries created by the (x, y, z) world system. Where each unit is called a block, and is exactly one meter in length, width, and height and is exactly sixteen pixels.

A connected hierarchical tree can be constructed at each z-level of the world, with transition points being single blocks that connect one z-level to the next. After this stage, three-dimensional RT-RRT\* will be able to function as it normally does in two-dimensions.

The machine utilized for this research is an MSI GT62VR 7RE with the following specifications:

* Windows 10 Professional
* Intel Core i7 7700HQ @ 2.8GHz
* 16 GB of DDR4-2400 Memory
* 480GB Intel 535 Solid State Drive
* NVIDIA GeForce GTX 1070 8GB

In addition, Project Malmo allows for writing agents in Java, C#, Python, and C++. However, for optimization and compatibility with all of Project Malmos’ features and NVIDIAs’ Compute Unified Device Architecture (CUDA) application programming interface, this thesis shall utilize the Visual C++ 2015 programming language.

1. **PLAN**

Preceding the actual implementation of the thesis, a few preliminary steps must be accomplished first, these include:

1. Install and configure Project Malmo on the development machine.
2. Create a data structure to represent the ellipsoidal tree of the search space.
3. Upscale RT-RRT\* to three dimensions.
4. Parallelize RT-RRT\* to work across multiple GPU cores.

After these steps are accomplished, the pathing algorithm will be hooked up to Project Malmo so that it may retrieve information from the game environment and send pathing information back to the agent playing the game. At this point, data collection can begin. In this stage, the pathing algorithm shall be run per the parameters outlined in the “Methodology” section. Once the algorithm is successful per the success criteria, it shall be allowed to run on several randomly generated worlds to gauge effectiveness (per the same pathing criteria for the predefined worlds) in times of true uncertainty. Results from testing on the predefined worlds shall be used as a baseline shall be used to gauge how well the agent performs on the random worlds.

This thesis will produce a three-dimensional version of real-time rapidly exploring random trees. It shall utilize Project Malmo as a testbed. To verify the algorithm, it will utilize several predefined, user-created worlds to ensure compatibility with the Minecraft environment. After this point, the algorithm shall be utilized on several randomly generated worlds. To ensure repeatability, the random world generated will be seeded with known Minecraft seeds for popular Minecraft worlds.

1. **PUBLICATION POSSIBLITIES**

This thesis shall be publishable in various journals and conferences that deal with artificial intelligence. Several publication possibilities include:

1. Journal of Artificial Intelligence Research
2. The Annual Symposium on Computational Geometry
3. The International Conference on Motion in Games
4. Artificial Intelligence: An International Journal
5. AI & Society: Journal of Knowledge, Culture, and Communication
6. Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies

# References

A. Ramires Fernandes and L. Deusdado. 2006. Efficient conservative collision detection for populated virtual worlds. In Ibero-American Symposium on Computer Graphics – SIACG(06), Santiago de Compostela, Spain.

Carlo Camporesi and Marcelo Kallmann. 2014. Computing shortest path maps with GPU shaders. In Proceedings of the Seventh International Conference on Motion in Games (MIG '14). ACM, New York, NY, USA, 97-102. DOI=http://dx.doi.org/10.1145/2668064.2668092

Joseph S. B. Mitchell and Micha Sharir. 2004. New results on shortest paths in three dimensions. In Proceedings of the twentieth annual symposium on Computational geometry (SCG '04). ACM, New York, NY, USA, 124-133. DOI=http://dx.doi.org/10.1145/997817.997839

Kenneth E. Hoff, III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH '99). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 277-286. DOI=http://dx.doi.org/10.1145/311535.311567

Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. 2015. RT-RRT\*: a real-time path planning algorithm based on RRT\*. In Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (MIG '15). ACM, New York, NY, USA, 113-118. DOI: https://doi.org/10.1145/2822013.2822036

Leonel Deusdado, A. Ramires Fernandes, and Orlando Belo. 2008. Path planning for complex 3D multilevel environments. In Proceedings of the 24th Spring Conference on Computer Graphics (SCCG '08). ACM, New York, NY, USA, 187-194. DOI=10.1145/1921264.1921302 http://doi.acm.org/10.1145/1921264.1921302

Michael Burch and Daniel Weiskopf. 2013. The aesthetics of rapidly-exploring random trees. In Proceedings of the Symposium on Computational Aesthetics (CAE '13), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 45-52. DOI=10.1145/2487276.2487285 http://doi.acm.org/10.1145/2487276.2487285