**Chapter 2: Literature Review**

CHRISTEN FORD, Baldwin Wallace University

# 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.

# 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.

# 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