19

Creating High-Order Navigation Meshes through Iterative Wavefront Edge Expansions

D. Hunter Hale and G. Michael Youngblood

19.1 Introduction

When placing AI-driven characters into your immersive game world, one large problem needs to be addressed, and that is the issue of a meaningful representation of the environment. The only source for information about the layout of the environment available to these characters is that which is provided to them by the game designers usually in the form of the geometric models that are assembled spatially to create the world. In all but the simplest of games, the level of detail in those model files is often too complex, too detailed, and organized more for display than spatial reasoning. Instead, some form of spatial abstraction is needed to group similar areas in single regions of space for the character to consider.

Historically, this representation was generally presented in the form of a waypoint map (i.e., valid points of known open points in a space with a collection of known good routes between them). Searching such a structure allowed AI characters to make paths through traversable space that appeared reasonable [Tozour 04]. The usage of waypoint graphs has been in decline as the navigation mesh spatial representation has risen in usage [McAnils 08]. A navigation mesh (often referred to as a *navmesh*) is composed of a listing of regions, which

are well-defined convex groupings of traversable space (usually defined by polygons or polyhedrons) and an additional listing describing connectivity (as a topological graph). This collection of regions organized as a graph can be rapidly searched to generate a path and characters can walk from region to region knowing they will remain in traversable areas.

Traditionally, navigation meshes have been created either by hand or using some form of automated spatial decomposition algorithm that examines the obstructions present in the environment and then breaks down the area between them into as few regions as possible. Reducing the number of regions present in a world yields a smaller search space and is generally considered to be highly important to a spatial decomposition. Unfortunately, creating a decomposition for a game environment with an optimal (absolute minimum) number of regions is NP-Hard [Lingas 82]. This means that there is no *best* technique. Instead there are many techniques that attempt to approach the optimal one. These approaches generally start with some form of triangulation of the environment [Delaunay 34] and then attempt to minimize the number of regions present in the environment through combining these triangles [Hertel 83].

The problem with this approach is that the triangles that remain in the navigation mesh cause problems for character navigation in areas where many triangles come together at a single point. It is all but impossible for a character to say which region they are standing in at the confluence points. These confluence points unfortunately show up all too often in complex environments. This leads to localization and pathfinding issues (i.e., if the character does not know where it is, then how can it find a path to its destination) [Hale 11].

The alternative to the triangular decomposition approaches, and one that will help minimize the character localization problem, is a growth-based approach. In our previous work we have presented 2D (PASFV) and 3D (VASFV) growth-based spatial decomposition algorithms [Hale 08, Hale 09], which were inspired by the Space-Filling Volumes algorithm [Tozour 04]. While these approaches do generate quality navigation meshes they can be slow when executed on large environments (the runtime of the algorithm increases based on the area to be decomposed). This is due to the fact these algorithms perform many unnecessary collision tests since they have to verify that every growing region has not intruded into another region or obstruction on every growth step. The vast majority of the time this is not the case, and this test will return a negative result. This unnecessary testing is a consequence of the sequential iterative expansion in traditional growth-based algorithms.

We have developed the Iterative Wavefront Edge Expansion Cell Decomposition (referred to as *Wavefront* for brevity) algorithm to address the problems of previous techniques by reducing collision tests and iterative growth. This algorithm works by scanning the world geometry visible from each region we place in the world and determines where possible collisions might occur (i.e., interesting places to expand toward). By forcing our regions to expand directly to these locations, we eliminate all but a handful of collision tests. This alters the runtime of the growth-based algorithm such that it increases with the complexity of the world (number of obstructions) instead of the area of the world. Not only is this technique faster than existing growth-based techniques, but the resulting navigation meshes produced using the Wavefront algorithm retain the high mesh quality exhibited by the PASFV and VASFV algorithms by providing regions of higher-order polygonal/polyhedron geometry [Hale 11].

19.2 Wavefront Spatial Decomposition

The Wavefront Edge Expansion Cell Decomposition (Wavefront) algorithm is derived from the PASFV and VASFV algorithms [Hale 08, Hale 09] and shares several implementation steps with them. The algorithm generates decompositions via a four-step process. First, unit-sized potential regions (seeds) are placed into the world. Next, one of these regions is selected at random and obstructions present in the world are analyzed from this region's perspective. In the third step of the algorithm the selected region enters a phase of accelerated expansion. This expansion is towards the obstructions found by the analysis in the second step of the algorithm. Steps two and three of the algorithm repeat for each region; this expands each region to their maximum possible size. Finally, in the fourth step of the algorithm new seeds are placed into any traversable space (a.k.a., empty space, unconfigured space, negative space) adjacent to the regions just created and the algorithm returns to step two, allowing these new regions to expand. If no new seeds are placed the algorithm terminates.

19.2.1 Initial Seeding

Traditionally, growth-based algorithms start using a grid-based pattern to place the initial unit-sized regions into the world. These approaches then iteratively give every region the chance to grow and expand outward in the direction of the normal of each edge (or face in 3D—we will use edge for simplicity here since they are both effectively boundaries for occupied space) of that region.

When using the Wavefront algorithm on our initial entry into the seeding phase we generate a list of *potential* seed points using a seeding algorithm that places a potential seed next to every exposed obstruction edge. This results in better overall coverage of the environment with fewer unit-sized quad (or cube in 3D) regions placed into the world over simple grid seeding [Hale 11]. Then one of these seed points is randomly selected to use as our initial region. The other potential seed points will be retained for later seeding passes, but will only be used if they are still in areas of traversable space that are as yet unclaimed by any regions. If on later passes through the seeding phase this list is empty, we will attempt to refill it by looking for areas of unclaimed traversable space adjacent to the regions we have placed. If this list remains empty after that point then the Wavefront algorithm will terminate.

19.2.2 Edge Classification

After a seed region has been generated, we proceed to the edge classification step of the Wavefront algorithm. These next two steps are the most computationally intensive steps of this algorithm, and we only wish to perform them on valid regions that we know are going to expand. Therefore, we only expand one region at a time and discard region seeds that are covered by earlier expansion. During this step, we iterate through each of the edges of obstructions present in the world as well as any edges present in regions that we have already placed into the environment. We then discard any edges whose normal faces away from the target seed point of the region as these edges are back facing and they cannot interact with the region. We then sort these edges into categories based on their relative spatial position when compared to the target seed location $(+x, -x, +y, -y, +z, -z)$. Note that this technique creates axis-aligned edges between decomposed regions (obviously

not guaranteed between regions to obstructions), which makes it easier for AI characters to reason and traverse regions. Edges that span multiple categories are placed in the first applicable one, depending on the evaluation order used in the implementation of the algorithm. Our reference implementation uses the following ordering *+y, -y, +x,-x, +z,* and *-z*. Any ordering will work as long as it is consistently followed.

Once the edges have been sorted, we locate all potential event points. Our region will have an edge that is perpendicular to each of the sorting classifications and whose normal matches the sorting classification (we will refer to this as the classification edge). By comparing the slope of each of our sorted obstruction edges to the appropriate classification edge, we can determine in advance how the expanding region would interact with the obstruction. This can be visualized by thinking of a radial half-plane sweep drawn from the initial seed point and then rotated in 90 degree arcs along each edge as shown in Figures 19.1 and 19.2. This sweep line will report the orientation of the edges it finds as well as the closest point on the edge to the initial seed point. The interactions between these edges of occupied space and the edge of the region we just placed can be reduced down to a series of cases.

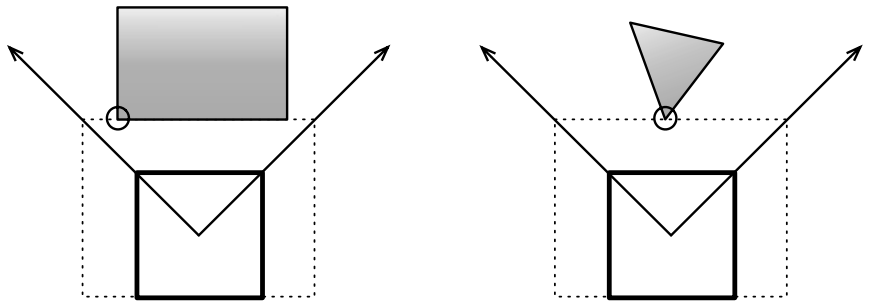

Figure 19.1

Two simple cases for event-based spatial decompositions: the case on the left shows expansion towards a parallel element and the case on the right shows the discovery of an intruding vertex.

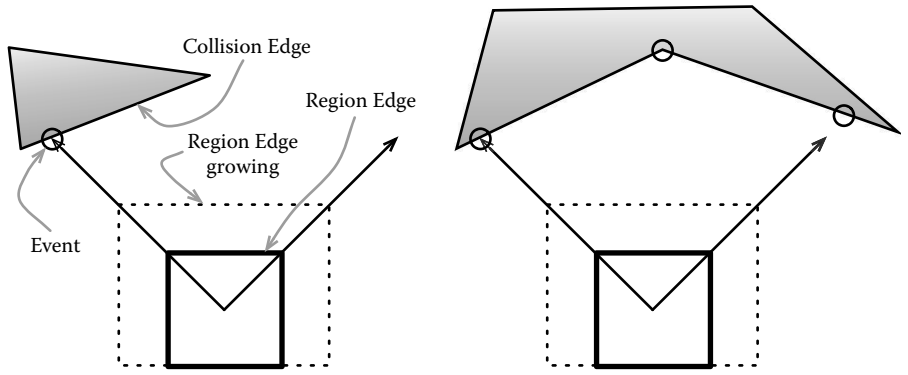

Figure 19.2

Two complex cases for the Wavefront decomposition involving splitting events.

The first of these cases occurs if the tested edge is found to be parallel to the classification edge as shown on the left side of Figure 19.1. In this case, we will wish to move the classification edge such that it is adjacent and co-planar to the target edge. We accomplish this by calculating the closest point on the edge to the initial seed point of the region we are evaluating. We then log this point and the distance from it to the region's initial seed point as an event. Incidentally, since all of our placed regions only expose axis-aligned edges and our expanding regions also only expose axis-aligned edges, any events involving other regions of traversable space will fall into this category.

A slightly more complicated case occurs when the edge is examined and found to be sloping inward towards the classification edge (i.e., region edge under evaluation) as shown in the right side of Figure 19.1. In this case, we will only be able to expand such that the closest vertex of that edge lies on the classification edge without changing the slope of the classification edge. We cannot change this slope as this would result in previously claimed areas of traversable space being relinquished, which would violate one of our invariants (claimed region space must always remain claimed by that region). This case is also resolved by storing the location of the closest vertex on the edge under evaluation along with the distance to that vertex from the initial seed point of the region.

Finally, we come to the most complicated case, which might result in the potential addition of new edges to the expanding region. In this case, as shown on the left side of Figure 19.2, the closest obstruction edge is sloping away from the midpoint of the classification edge, and it would be possible to move the classification edge such that one of its vertices could intersect the edge under consideration. This is an edge splitting case, and in order to calculate where this split should occur, the closest point on the edge under evaluation to the initial seed point of the region is found. This point is then stored as an event point along with the distance between this point and initial seed point of the region. Additionally, we wish to store the two end points of the edge under consideration (assuming the closest point was not an end point) so that we will be able to increase the order (order indicates the relative number of sides of a polygon/polyhedron, so a triangle has order three, an octagon order eight, and so forth) of this region such that it adds a new edge that is adjacent to the entire length of the edge under consideration. However, instead of calculating the distance between each of these end points and the initial seed point, we will treat them as a special case that looks at them as if they are only slightly further away from the initial point than the point we are using to split. This will prevent those points from interfering with other calculations in the process.

A more complex case with multiple splitting events can be seen on the right side of Figure 19.2. The events should be processed in order based on the distance from the initial seed point of the expanding region, and by altering the distance of these two end points we will ensure that the region tries to fully encompass all of the space that is adjacent to the edge it splits on as that point is processed.

At this point we have a collection of potential events for our new region to expand towards; however, we need to do two things before we can begin the expansion. First, if the edges of the world are defined as some boundary conditions rather than nontraversable space, events will need to be inserted to allow each region to expand outward to the edges of the world. Then this list will need to be sorted based on the distance between each event and the initial seed point of the region. This results in the processing of closer events first

as we are more likely to reach them as further events are oftentimes unreachable due to the presence of more immediate obstructions.

19.2.3 Edge Expansion

With the completed event list for this region we are able to proceed to the expansion phase of the Wavefront algorithm. First, the expansion rates of all of the edges of the region are reset to zero. Then, the first (closest) unprocessed expansion event is selected and removed from the list of potential events. The distances that the edges of the region would have to move such that they reach this expansion event are then calculated. This is done by calculating the distance between the current location of the two (three in 3D) closest edges and the target expansion location. This result is then broken down into its principal components (x, y, z) and if these values are positive they are set as expansion rates for the edge or edges that have a normal that points toward the target event. The use of rates is a legacy from stepwise growth, but here the rates indicate jumps directly to event points. Expansion should then occur with each edge iteratively moving outward. Once all the edges have moved, then the check for any collisions or invalid expansion conditions can be executed. This happens because there are splitting events that may result in invalid configurations if only half of the event (i.e., one rather than two edges are allowed to expand) is executed.

Once the region has finished expanding, any collisions with other regions or obstructions must be resolved. Any vertices of the expanded region that collided with an obstruction must be split, and the region must be converted to a higher-order polygon/polyhedron by inserting a new edge. To construct this new edge take the opposite normal of the obstruction edge and constrain this new edge to the extents of the obstruction edge. Since expansion events are calculated in isolation with no consideration for other regions or potential obstructions, it is possible that a collision will occur and that the region will have to contract from a potential expansion event. If this happens, then the edge involved in the collision should cease further attempts to expand. The algorithm will then select another expansion event, repeating this process until there are no more events or all of its edges have ceased attempting to expand due to collisions.

19.2.4 Reseeding

After all regions have finished expanding, additional regions will be placed as per the seeding process discussion earlier. If the algorithm enters the seeding phase, and is unable to place any new regions, it terminates. This results in a collection of regions that is ready to serve as a navigation mesh. Additionally, if desired, this collection of regions can be cleaned up by combining adjacent regions such that the result would still be convex.

19.3 Postdecomposition

Existing growth-based spatial decomposition algorithms (e.g., PASFV, VASFV, and SFV) took advantage of a postprocessing step to improve the quality of the resulting navigation mesh. Occasionally, two or more region seeds will grow into an area of the environment that could be filled by a single convex region. This is a natural consequence of placing and growing multiple seeds at the same time, and is generally corrected by combining the regions. However, this combining takes time and effort, and it would be nice if it was

not required. A strength of the Wavefront algorithm is that it avoids most of this form of cleanup due to the fact it only grows one region at a time. Since two regions are never growing at the same time, they cannot both attempt to subdivide the same convex area of traversable space, thus yielding a cleaner decomposition.

19.4 Wavefront Runtime

The Wavefront algorithm enjoys a worst case runtime, bounded by the complexity of the environment it is executed on of $O(n*m)$. In this case n is the number of obstructions present in the world, each of which will have to be evaluated by m regions that will be seeded by the algorithm. This runtime might seem to be worse than existing growth-based spatial decomposition algorithms (they generally increase fractionally, $O(n^{1/x})$ where n is the number of square units in the world, and x is the number of regions), but remember that the runtimes of these increase based on the size of the world (due to the additional growth steps that have to be performed to fill the world).

The runtime of the Wavefront algorithm only increases with the actual complexity of the environment and not due to the introduction of additional unoccupied space. In general, across a variety of game environments of different sizes and complexities, our reference implementations of these two algorithms average runtimes in the milliseconds to seconds range for Wavefront in comparison to a range of seconds to minutes for our growth-based implementations. The memory footprint of the Wavefront algorithm grows linearly as each newly generated region only needs to interact and know about existing regions and obstructions at any given point in time.

19.5 Comparisons to Existing Techniques

The Wavefront algorithm has been compared to existing methods of generating spatial decompositions with particular focus on those currently in use in industry, namely Delaunay Triangulation, Hertel–Melhlorn Decompositions, and Trapezoidal Cellular Decompositions. We only targeted algorithms for comparison that also generate full coverage decompositions in order to ensure the comparisons were valid. Evaluations were conducted on 25 procedurally generated worlds composed of randomly generated and placed obstructions with no axis-aligned restrictions and a basic set of rules that generated test worlds similar in geometry to those found in many games (the generation rules were influenced from public *Quake 3* levels, which were used in initial testing).

We generated decompositions for the worlds using each algorithm under consideration (one of these levels and the Wavefront Decomposition for it is shown in Figure 19.3). We then evaluated the decompositions based on the number of regions present, and the quality of the decomposition (using navigation mesh evaluation metrics [Hale 11] to determine the number and shape of any degenerate or low quality regions). We found that the decompositions generated with the Wavefront algorithm contained both fewer total regions and fewer near-degenerate regions than the Trapezoidal Decompositions or Delaunay Triangulation Decompositions. We define a near-degenerate region to be one that an AI character would have difficulty moving into or out of. Such regions are characterized as oddly or bizarrely shaped areas (e.g., fans of triangles all coming together at a single point, long thin slivers of quad-based regions spanning an environment, regions

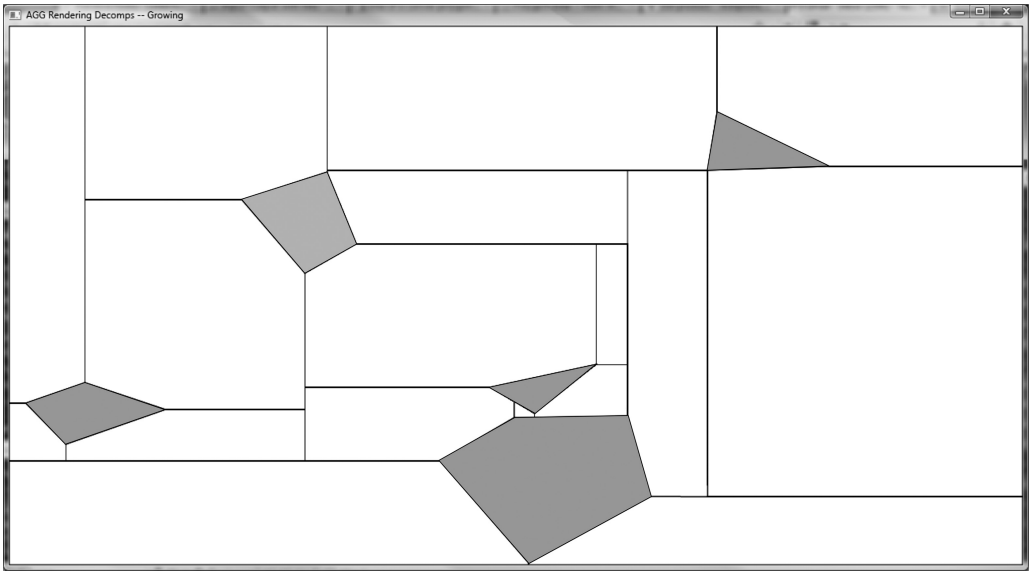

Figure 19.3

A decomposition produced by the Wavefront algorithm. Obstructions are shown in gray, while the decomposition regions are shown with black outlines.

with very narrow adjacencies to other regions, or disjoint/poorly connected regions), which we are able to test for and detect.

The number of regions was consistent between the Wavefront algorithm and the Hertel–Melhlorn decomposition; however, the Wavefront decompositions had fewer near-degenerate regions than the Hertel–Melhlorn decompositions [Hale 11]. It is not surprising that the Wavefront algorithm has fewer of these near-degenerate regions as it possesses a unique property not shared by the other decomposition techniques we tested against. Namely, there is an upper limit on the number of regions that come together at a single point of traversable space of five (10 in 3D); mathematical proof in [Hale 11]. The other commonly used techniques have no upper bound on how many regions can converge at a single point of traversable space. This convergence of many regions onto a single point is what often leads to the creation of near-degenerate regions and should be avoided if possible.

It is worth noting that the Wavefront algorithm generates decompositions that appear to be similar to those generated by the Trapezoidal cell decomposition algorithm. However, they are distinct decompositions, due to the fact the Wavefront algorithm will consistently produce decompositions with fewer regions. This is due to Trapezoidal Decomposition being restricted in only decomposing the world in a single direction (vertical or horizontal) while the Wavefront algorithm is in effect a multidirectional decomposition (both vertical and horizontal wavefronts originate from the initial region seeds).

For detailed information on the evaluation of the Wavefront technique, quantitative numbers, and navmesh quality metrics, please refer to Hale's *A Growth-Based Approach to the Automatic Generation of Navigation Meshes* [Hale 11].

19.6 Conclusion

Overall, the Wavefront algorithm generates fast, high-quality decompositions for use as navigation meshes via a quad-based expansion algorithm. Such decompositions have fewer small and degenerate regions (generally triangles) that can interfere with character navigation. This algorithm improves on previous growth-based approaches by performing fewer expansion steps, which reduces the number of collision tests that must be performed. This yields an algorithm whose runtime scales with the complexity of the world rather than the size of the world as existing growth-based approaches do. Additionally, since this algorithm only grows one region at a time, there is less post processing that would normally be caused by multiple regions competing to fill the same convex area. The decompositions generated by this algorithm compare favorably with those produced by existing popular algorithms (e.g., Hertel–Melhlorn or Trapezoidal Cell Decomposition).

References

[Delaunay 34] B. Delaunay. "Sur la sphere vide" *Classe des Sciences Mathematiques et Naturelle* 7. 1934.

[Hale 08] D. Hunter Hale, G. Michael Youngblood, and P. Dixit. "Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds." *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2008.

[Hale 09] D. Hunter Hale and G. Michael Youngblood. "Full 3D spatial decomposition for the generation of navigation meshes." *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2009.

[Hale 11] D. Hunter Hale. A Growth-Based Approach to the Automatic Generation of Navigation Meshes. Doctoral Dissertation. University of North Carolina at Charlotte, December 2011.

[Hertel 83] S. Hertel and K. Mehlhorn. " Fast triangulation of the plane with respect to simple polygons." *International Conference on Foundations of Computation Theory*. 1983.

[Lingas 82] A. Lingas. "The power of non-rectilinear holes." *Proceedings 9th International Colloquium on Automata, Language, and Programming*. 1982.

[McAnils 08] C. McAnils and J. Stewart. "Intrinsic detail in navigation mesh generation." In *AI Game Programming Wisdom 4*. Hingham, MA: Charles River Media, 2008, pp. 95–112.

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