Dynamic Updating of Navigation Meshes in Response to Changes in a Game World

D. Hunter Hale and G. Michael Youngblood
The University of North Carolina at Charlotte
Department of Computer Science
9201 University City Blvd, Charlotte, NC 28223-0001
{dhhale, youngbld}@uncc.edu

Abstract

We present a modified navigation mesh generation algorithm that allows the mesh to be dynamically altered at runtime. We accomplish this using an extension to the existing spatial decomposition algorithm ASFV (Adaptive Space Filling Volumes) that will allow the algorithm to dynamically adapt to changes to the underlying world geometry without having to rebuild the entire spatial decomposition. This is accomplished by providing two algorithms to deal with alterations to the world. The ability is provided to add arbitrary obstructions into what was negative space and then to build a new correct spatial decomposition around the new obstruction. Functionality is also provided to remove existing obstructions and then to build up new decompositions to fill in the newly created negative space. Finally, we show via an experiment that our dynamic extensions to ASFV reduces the cost of correcting an invalidated decomposition by 90% or more.

Introduction

When using an embodied agent in virtual worlds one of the first questions that must be answered is how the world will be presented to the agent. When constructing a 3D environment for a game or simulation, usually the only areas that are well defined are the 3D models or objects included in the world. Rendering the world to the screen generally requires the creation of a nicely delineated listing of the occupied or positive space objects present in the world. In general, this information is made available to the agent, which provides the agent with minimal information about the world. Unfortunately, this listing does not provide any information about the open empty areas of the world that we will call negative space.

Negative space can be presented to the agent in several different ways. All the methods discussed here will simplify the 3D world to a 2D footprint. This is similar to the manner in which architects simplify a building to its floor plan when creating blueprints. About half of the available methods for evaluating negative space focus on presenting a set group of paths to the agent to aid in getting from point A to point B and can be referred to as way-pointing methods. This works reasonably well for providing a resource to the agent to aid in path planning, but does not really help the agent break down areas of negative space in order to reason only about relevant and local events to the the agent. Other methods of presenting negative space to an agent work by dividing the world in to some form of discrete regions. This is referred to as decomposing the world or a spatial decomposition, and the collection of discrete regions can be called navigation meshes, navigation maps, or spatial decompositions (Tozour 2004). The ASFV (Adaptive Space Filling Volumes) algorithm (Hale, Youngblood, & Dixit 2008) for which we will present an extension is a spatial decomposition algorithm. Once the world has been decomposed, a graph can be constructed that shows connectivity between the regions. Such a graph provides an excellent tool for the agents to reason about path planning using any search algorithm the developer chooses to implement.

All the methods of evaluating negative space we have discussed so far have one rather disappointing thing in common, the end product of each tends to be very static. This is unfortunate as there is a tendency towards more interaction with the world in modern games and a desire for more realistic reactions from the environment in simulation. In several recent game releases it has been possible to dynamically alter the world in response to the players actions. For example, in the recent title from Electronic Arts *Battlefield: Bad Company* it is possible for the player to demolish walls in structures and dramatically alter the passable areas of the game world. The title *Fracture* from Lucas Arts carries this concept of a deformable world further as it allows the player to dynamically alter the terrain of the world. Obviously, if the player is given the total freedom to alter the world it will be impossible to predict the state of the world at any given point in the game and the existing solution of having a pre-built spatial decomposition or set of way points for every possible world state cannot hope to adapt to this level of variability. In response, there is a need to upgrade and expand the spatial decomposition techniques in order to deal with this random alteration of the world.

We propose a solution for this problem that involves an extension of the existing algorithm Adaptive Space Filling Volumes in order to allow it to dynamically rebuild and relink damaged areas, such that even with dramatic changes to the world the decomposition is still valid. This extension comes in two distinct parts. The first part allows for

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

addition of negative space into regions that were formerly positive space obstructions (or the removal of positive space regions). Imagine a helicopter takes off leaving an empty field, this field can now be walked through freely and the decomposition will need to be updated to reflect this. The second extension allows for addition of positive space regions into areas that were formerly walkable. For example, imagine a building collapsing into a plaza area. This will render the plaza partially or completely unnavigable and obviously require an extensive change to the navigation maps. Both of these extensions are natural progressions from the core concepts of the ASFV algorithm which are: providing a fast effective way of generating a world decomposition, providing a high coverage decomposition, yielding good quality decompositions for navigation, and providing decompositions based around quads or higher order polygons rather than triangles allowing for better information compartmentalization.

Related Work

Space Filling Volumes (SFV) is the parent technique of our Adaptive Space Filling Volume technique (Tozour 2004). It does provide a decomposition of the world on which a navigation mesh could be built, but it tends to provide less than optimal decompositions. SFV works by seeding the world initially with geometric quads and growing them until they hit an obstruction. This works well for worlds composed solely of axis aligned rectangular obstructions, but on worlds containing arbitrary obstructions SFV tends to leave large gaps of un-decomposed space. These gaps can force agents to take longer paths to traverse the world and large areas of the world could potentially be unreachable. Since SFV is the parent technique of the ASFV the extensions to ASFV we present here would be applicable to it as well.

Navigation Meshes can also be generated via the Hertel-Mehlhorn (Hertel & Mehlhorn 1983) algorithm or with Delaunay Triangulations (de Berg *et al.* 1998). Both these methods focus on connecting vertices of obstructing geometry to generate a collection of regions. Hertel-Mehlhorn focuses on generating the minimum number of regions that provides complete coverage and can produce high order polygons, while Delaunay Triangulations produce a purely triangular decomposition with a focus on producing the most uniform triangles possible. Both these algorithms generate excellent coverage decompositions that work well for navigation, but can create problem areas of very small triangles that cause problems with localizing objects to a single area. Level designers also conduct by-hand decompositions that loosely adhere to one of these two algorithms to generate specific decompositions that better reflect the intended uses of the world. Obviously, by-hand decompositions cannot be used to dynamically update navigation meshes and these other two algorithms take too long to rebuild a decomposition in response to change in the world. However, the extensions presented to ASFV could be applied to initial decompositions from by-hand decompositions, Hertel-Mehlhorn, or Delaunay Triangulations to properly adjust to dynamic worlds.

Recently, work has been conducted to create navigation meshes using a rendering based approach called Render-Generate (Axelrod 2008). This approach works by iteratively rendering depth maps of the world, and using these maps to calculate the locations of the floors and ceilings along with the positions of any obstacles. Using the slopes and obstructions present in these depth maps it is possible to find areas the agent is capable of standing inside. By connecting adjacent standable areas a walkability map can be generated. This algorithm is capable of decomposing a complex environment in seconds.However, decompositions generated by this algorithm are limited to constant cell sizes, which are generally defined as the size of the agent to navigate the world, and no simplification is done on the resulting graph. This tends to produce graphs in which relatively small areas have a large number of regions, which slows down most path finding algorithms.

There are many non-spatial decomposition forms of generating simple navigation data such as Probablistic Roadmaps, Voroni Graphs, Waypoints, and so forth (Russell & Norvig 2003). Work has been conducted to allow these systems to dynamically adapt to changes in the environment. Most of this work is focused on ways to reduce the number of regions that need to be replaced, and on limiting the amount of geometry that needs to be considered during the rebuild. A good example of this is the technique presented by Marden and Smith (Marden & Smith 2008) for minimizing changes to a waypoint system due to dynamic objects. However, none of these provide useful secondary services such as information compartmentalization or collision detection.

Methodology

The primary contribution of this paper is an extension for the existing ASFV algorithm that allows it to dynamically adapt to changes in the underlying geometry of the world at run time. This extension is provided by two algorithms that allow for the addition of negative space into existing positive space locations or the addition of positive space into existing negative space. We refer to the enhanced version of ASFV as Dynamic Adaptive Space Filling Volumes (DASFV).

Adaptive Space Filling Volumes

The Adaptive Space Filling Volumes (ASFV) algorithm, which serves as the basis of our dynamic decomposition repair system, is a straight forward algorithm. First, the ASFV calls for the placement of a grid of unit sized quads into the world to be decomposed. These quads then provided an iterative chance to expand in every direction. At this point, the algorithm is very similar to the classic SFV algorithm we discussed earlier and the algorithm draws its inspiration from that work. The first difference and reason for the *Adaptive* in this algorithm's name occurs when one of the growing vertices of a quad hits a piece of obstructing geometry. Unlike Space Filling Volumes, when ASFV encounters an obstruction it has the ability to dynamically increase the order of its growing quads into more complex polygons—though it will ensure that each polygon is still convex. After all the polygons have grown to the maximum possible extent, the

algorithm will attempt to reseed the world with more unit sized quads that are then provided with the ability to grow. This cycle of grow and seed continues until no more seeds can be placed at which point the world will be fully decomposed. Using this algorithm it is possible to generate a high quality decomposition of the environment that can be applied to multiple uses in a game or simulation world

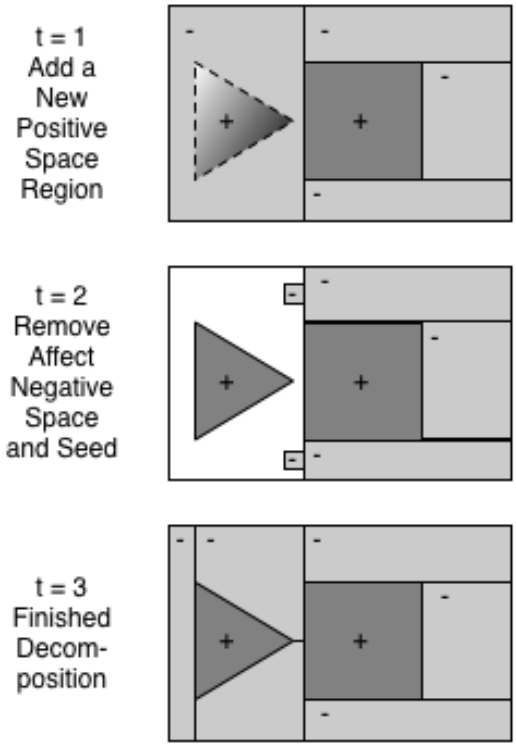

Figure 1: A time step progression of the addition of positive space. The negative spaces are shown in light grey. Positive spaces are shown in dark grey. The positive space that is added to the world is shown with a gradient and it has the dotted outline. The second time step shows the removal of affected negative space regions and the addition of seeds to grow new negative space regions. The final time step shows the repaired decomposition.

Using this algorithm it is possible to generate spatial decompositions that can be shown experimentally to produce better navigation paths that other spatial decomposition techniques such as Hertel-Mehlhorn or conventional Space Filling Volumes (Hale, Youngblood, & Dixit 2008). In addition, this algorithm can be shown to run in time $O(n^c)$ where c is between 0 and 1 (typically 0.5) and n represents the size of the area in question to be decomposed (Hale & Youngblood 2009). More complex geometry tends to increase c while a denser initial seeding tends to decrease it.

Adding Positive Space

Let us first examine the case where the building collapses into the plaza. First, the negative space regions that used to compose the plaza need to be removed. These are found by performing a series of intersection tests on the existing negative space regions and the newly added region. Since all of the negative space regions are convex and the newly added positive space regions are required to be convex, the intersection test between them can be performed very quickly with a point in convex polygon algorithm (Schneider & Eberly 1998). Once a list of negative spaces to be removed has been established, the algorithm will branch into two directions depending on what information is available. If connectivity information is available between regions then the neighbors of the regions to be removed should be reset so that they will attempt to seed as per the ASFV algorithm. If connectivity information is not available then all negative space regions not being removed should be reset to seed again. At this point the ASFV algorithm will be started at the seeding stage of the algorithm. ASFV will run until the newly changed region is fully decomposed at which point it will stop as per the original algorithm. Sample code is provided for this case in Algorithm 1 and an illustration of this algorithm in action is provided in Figure 1.

Algorithm 1 Place Positive Space

```
//We will assume that the area was fully decomposed
HandlePositiveAddition(List oldNegatives, List oldPositives,
newPositives)
//Locate all areas that need to be removed
for all NegativeSpaces neg in oldNegatives do
   //Run intersection check against each new region
   for all PositiveSpaces pos in newPositives do
      if neg.intersects(pos) then
         neg.remove()
         if connectivityKnown is true then
            //set adjacent regions to seed
            neg.setNeighborsSeeding(true)
            //otherwise do nothing
         end if
      end if
   end for
   if connectivityKnown is false then
      neg.ResetSeeding()
   end if
end for
//All conflicting regions will have been removed
//Assume the implementation of ASFV contains
//a method to start seeding
ASFV.seed()
//Also assume that the ASFV implementation
//contains a method to grow placed seeds
ASFV.run()
//Once the ASFV algorithm concludes the decomposition
//will be complete again
```

Adding Negative Space

The addition of negative space regions works similar to the addition of positive space regions. To continue the example of the helicopter taking off from a field we need to quickly determine how to divide up the field it vacates for navigation. First, the listing of all of the positive space regions to be removed is compiled. Once this has been compiled the algorithm will again branch depending on whether or not

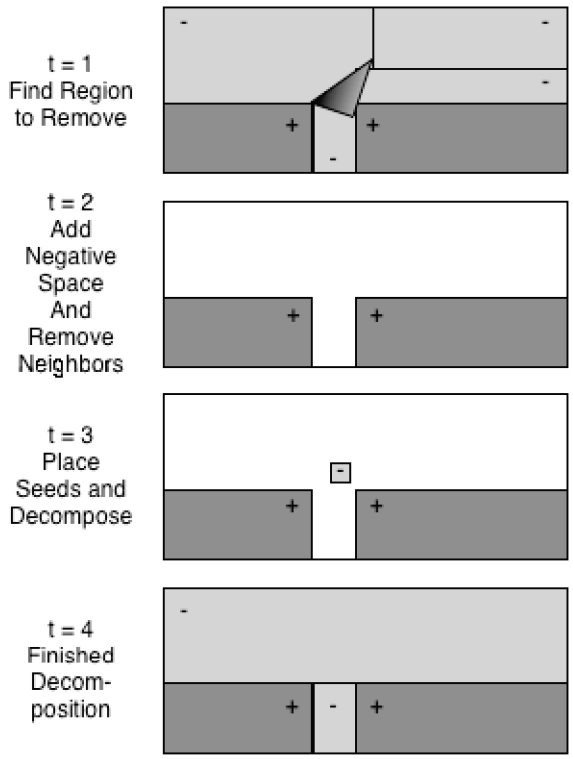

Figure 2: A time step progression of the addition of negative space. The negative spaces are shown in light grey. Positive spaces are shown in dark grey. The positive space to be removed is marked in a gradient. In the second time step the targeted positive space has been removed along with its neighboring negative spaces. In the third time step a seed has been added to the decomposition. In the final time step the decomposition has been fully repaired.

connectivity information is available. If connectivity information is known between regions then the neighbors of the affected positive space regions can easily be located. Otherwise the neighbors have to be located algorithmically. This is accomplished by taking the affected positive space regions increasing their size by 0.1 percent, and then determining what negative space regions they intersect. Since the decomposed regions perfectly border against the obstructions this small size increase to a positive space region will force it to intersect all their neighboring negative space regions. Once the targeted positive space regions and neighboring negative space regions have been identified they can then be removed. Seeds are then placed at the former centroids of the removed positive space regions to provide a starting point for the re-initialization of the ASFV algorithm. Again, once ASFV completes the newly exposed free space will be fully decomposed into high order convex polygons. The presentation of this algorithm can be seen in Algorithm 2. An illustration of this algorithm running can be seen in Figure 2

In the case of the removal of a positive space region it may seem strange and counter intuitive to remove the negative space regions adjacent to it when they do not appear to be directly affected by the removal, but there are several reasons for doing so. The primary reason for removing neighboring negative space regions is related to the concept of information compartmentalization, so let us take a moment to consider how the shape of regions affects this quality. As long as we are just using these decompositions for agent navigation from point-to-point along some kind of connectivity graph and the decomposition completely covers the free space and is well connected, the ability of the decomposition to compartmentalize information is not of great importance to the agent. However, when we move beyond using the decompositions for just navigation we encounter issues with triangular based decompositions such as those generated using the Hertel-Mehlhorn algorithm (Hertel & Mehlhorn 1983) or Delaunay Triangulations (de Berg *et al.* 1998). Decompositions of environments can also be used for the encoding and storage of relevant spatial details for the game or simulation. For example, objects and events can be localized to regions. This means that the agent moving around the world can reduce the amount of things it needs to reason about to those objects in its region and perhaps neighboring regions. Localized dynamic object collision detection can also vastly reduce the amount of collision tests per frame by only considering possible collision cases for objects that occupy the agent's region or neighboring regions. It is in situations like this that the triangular decomposition methods begin to have problems. There is no limit to the number of triangles that can come together at a point so any given triangle could

Algorithm 2 Add negative space to existing positive space regions

```
//We will assume that the area was fully decomposed
removePositive(List oldNegatives, List oldPositives, List toRemove)
// Locate all areas that need to be removed
for all PositiveSpaces oldPos in oldPositives do
   // Run a simple equality check to find regions to remove
   for all PositiveSpaces posRemove in newPositives do
      if oldPos.equals(posRemove) then
         //Found a region to remove locate its neighbors
         //Assume we possess a function
         //to find neighbors of a given region
         List neighbors = oldPos.getNeighbors()
         for all NegativeSpace neighbor in neighbors do
            //Remove the negative space neighbor
            neighbor.remove()
         end for
         oldPos.remove()
      end if
   end for
end for
//All conflicting regions have been removed
//Assume the implementation of ASFV contains a
//method to start seeding
ASFV.seed()
//Also assume the ASFV implementation contains a
//method to grow placed seeds
ASFV.run()
//Once the ASFV algorithm concludes the
//decomposition will be finished
```

have a huge number of potential neighbors. Furthermore, since triangles by their nature tend to be long and skinny in these decompositions it is harder to say that objects can be contained in only one or two regions. Reasonably sized dynamic objects in a game might span the narrow edges of many triangles or even worse might sit at the convergence point of dozens of triangles. This high overlap potential can drastically reduce potential computational savings of using decompositions of the world to reduce and localize the number of expensive computations that must be performed.

By its very nature the ASFV algorithm can only produce negative space regions that intersect each other in two ways. The first intersection type occurs when axis aligned parallel edges meet each other, in this case there can obviously only be at most two regions involved in the collision. The other possibility occurs when the right angle corners of three or four growing regions meet at a single point. This in effect limits the number of regions meeting at a single point to four, which is obviously far lower than the unbounded worst case for triangular decomposition methods. As such, ASFV-based decompositions tend to lend themselves more towards applications beyond navigation which depend on good information compartmentalization. However, in order to ensure this is the case when we remove positive space regions we need to ensure that we do not expose non-axis aligned edges of any negative space regions. Doing so would potentially introduce all of the problems associated with allowing for the possible intersection of more than 4 regions at a single point. This is the primary reason that our algorithm calls for the removal of neighboring negative space areas when a positive space region is removed. Also, since information compartmentalization tends to work better on larger regions it benefits us to clear out any neighboring regions to ensure that we get the largest regions possible. The problems that could be caused by just removing positive space regions and not their negative space neighbors can be seen in Figure 3. Despite starting from the same initial configuration, the decomposition of the world in Figure 3 is is clearly worse than Figure 2 since it contains 5 malformed regions rather than 2 axis aligned regions.

After the decomposition of the world has been adapted to fit the changes in the world's underlying geometry the connectivity graph between regions can be quickly rebuilt. Instead of completely rebuilding the connectivity graph a shortcut using only two simple steps is available. The first step is to remove any links that are invalidated by the removal of negative space regions. Then it is a simple matter of iterating through each of the new regions that were created during the decomposition repair phase and determine which regions border it to rebuild the connectivity graph between each region.

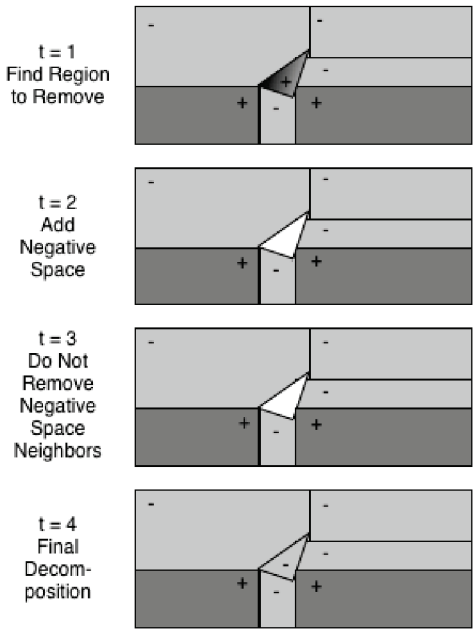

Figure 3: This timestep illustration shows the undesired effects of adding negative space to a positive space region without removing the neighboring negative space region as called for in the Dynamic Space Filling Volumes algorithm.

Experimentation

An experiment was conducted on both aspects of dynamically altering the world after a decomposition has been generated, those aspects being the addition of negative space, and the addition of positive space. Both experiments were conducted on the same computer using the DEACCON (Decomposition of Environments for Automated Creation of Convex Navigation-Meshes) tool which implements the DASFV algorithm on a 2.13Ghz/2GB RAM computer.

The first experiment consisted of examining the effects of adding positive space to existing areas of negative space. This is the building collapsing into the plaza example from above. This experiment was conducted on 10 random maps each with a increasing amounts of geometric complexity which represent other buildings that are present in the world. The performance of dynamically repairing the decomposition using our new extensions to Adaptive Space Filling Volumes is compared to the time it takes to completely rebuild the decomposition in order to adapt to changes. As can be seen in Figure 4 using our dynamic extension to generate new decompositions for altered environments is considerably faster compared to rebuilding the entire decomposition even as the complexity of the world in which the algorithm operates increases. These results are in accordance with our expectations that the rebuilding a smaller area of the decomposition is faster than rebuilding the entire decomposition.

The second experiment consisted of examining the effects of adding negative space to existing positive space regions, in effect removing those regions, and then decomposing the newly created free space. Our helicopter taking off example would fall in this experiment. Again, we compared the time cost to decompose just the affected areas to the cost of rebuilding the entire decomposition against a backdrop of increasing geometric complexity. The results of this experiment as shown in Figure 5 does generally show that it is better to rework just the affected area than the entire decomposition. However, there are some cases for very simple

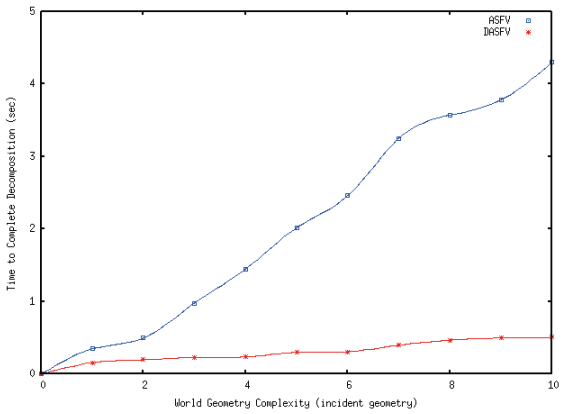

Figure 4: This graph shows a comparison between the time costs to repair a decomposition vs rebuilding the entire decomposition when positive space is added to the world while the complexity of the world is increased.

worlds containing only one or two regions where the repair takes as long or longer than rebuilding the entire decomposition. A close examination of the worlds in question yields an explanation of why this phenomenon occurs. Recall the addition of negative space algorithm calls for the removal of all neighboring areas of negative space in addition to the directly affected positive space region. This means for very simple worlds it is possible all or most of the negative space in the world is removed along with the positive space target. This results in the repair algorithm having to run intersection tests on every region in the world before discarding all of them. Obviously these intersection tests will take longer than just discarding everything to start with. However, such simple worlds occur so infrequently that is always desirable to perform a repair rather than a reconstruction.

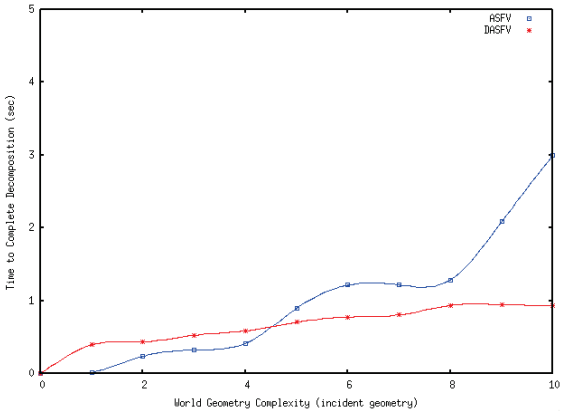

Figure 5: This graph shows a comparison between the time costs to repair a decomposition vs rebuilding the entire decomposition when negative space is added to the world while the complexity of the world is increased.

Conclusion

In conclusion, our Dynamic Adaptive Space Filling Volumes algorithm provides a fast and high quality way to regenerate invalidated spatial decomposition maps without having to rebuild them from scratch. We have shown experimentally that regeneration is in the vast majority of cases considerably faster than attempting to fully rebuild the world decomposition. Other decomposition methods lack this form of dynamically adapting to changes in the underlying world geometry and, in addition, previous work has shown the navigation meshes generated via the other are no better than ASFV in terms of agent navigation.

Acknowledgments

This material is based on research sponsored by the US Defense Advanced Research Projects Agency (DARPA). The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the US Government.

References

Axelrod, R. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.6 Navigation Graph Generation in Highly Dynamic Worlds, 125–141.

de Berg, M.; van Kreveld, M.; Overmars, M.; and Schwarzkopf, O. 1998. *Computational Geometry : Algorithms and Applications*. Springer.

Hale, D. H., and Youngblood, G. M. 2009. A Theoretical Analysis of the Runtime of ASFV. Technical Report GL2-UNCC-09-01, The Playground Lab, Computer Science, UNC Charlotte.

Hale, D. H.; Youngblood, G. M.; and Dixit, P. 2008. Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Hertel, S., and Mehlhorn, K. 1983. Fast Triangulation of the Plane with Respect to Simple Polygons. In *International Conference on Foundations of Computation Theory*.

Marden, P., and Smith, F. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.3 Dynamically Updating a Navigation Mesh via Efficient Polygon Subdivision, 83–94.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc.

Schneider, P. J., and Eberly, D. H. 1998. *Geometric Tools for Computer Graphics*. Morgan Kaufmann.

Tozour, P. 2004. *AI Game Programming Wisdom 2*. Charles River Media. chapter 2.1 Search Space Representations, 85–102.