Full 3D Spatial Decomposition for the Generation of Navigation Meshes

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

Abstract

We present a novel algorithm developed for decomposing world-space into arbitrary sided high-order polyhedrons for use as navmeshes or other techniques requiring 3D world spatial decomposition. The Adaptive Space Filling Volumes 3D (ASFV3D) algorithm works by seeding world-space with a series of unit cubes. Each cube is then provided with the opportunity to grow to its maximum extent before encountering an obstruction. ASFV3D implements an automatic subdividing system to convert cubes into higher-order polyhedrons while still maintaining the convex property. This allows for the generation of navigation meshes with high degrees of coverage while still allowing the use of large navigation regions—providing for easier agent navigation in virtual worlds. Compared to the Space-filling Volumes and Automatic Path Node Generation navigation mesh decomposition methods, ASFV3D provides more complete coverage and a less complex navigation mesh.

Introduction

Agents need information about the world in which they are operating in order to behave in a believable manner. One of the best and most commonly used methods to provide spatial environmental information to agents about the world is to create a mesh of convex shapes that represent all the areas in the virtual environment that the agent is capable of transversing (Tozour 2004). These *navigation meshes* provide the agent with a variety of useful information. In addition, there are several positive effects to having a high quality navigation mesh that can prove advantageous to the engine running the game. The most basic of these advantages is improved path finding capability provided to the agent by reducing the empty space where the agent can travel from many thousands or millions of points down to the dozens to hundreds of regions present in a navigation mesh. This leads to a runtime improvement for most pathfinding algorithms. Another application for spatial decomposition meshes is information compartmentalization, which reduces the number of objects an agent has to reason about and interact with to just those that reside in the same or neighboring spatial regions. This reduces overall reasoning complexity over objects. Collision detection can also be improved if an agent or other object can be quickly localized to a single walkable region since then they are not colliding with the environment and can only collide with other objects inside that one region.

Navigation meshes can be constructed in many different ways from the world geometry. However, most methods for building navigation meshes fall into one of two categories. The first and most commonly used set of techniques to generate a navigation mesh is by vertex-based decomposition. Using a set of rules, all of the vertices exposed by the world geometry are connected to each other to generate a series of triangles. These triangles can then be combined where the result would be a convex higher-order polygon to reduce the total number of shapes present in the navigation mesh. Vertex-based approaches generally give very high coverage decompositions, but can result in lots of small or strangely shaped regions, many of which can come together at a single point causing problems when attempting to localize objects to a single region (because most objects are larger than a single point in space). The other commonly used general approach to generate a navigation mesh is the growth-based method. Some form of geometry is sown into the world, then each piece of this geometry is allowed to expand until it hits an obstruction. These pieces of geometry are then connected where they touch and formed into a navigation mesh. Traditionally, growth-based methods have provided very regular-shaped regions, but have not provided high coverage breakdowns of the world. However, our prior work has presented improved methods to decompose open space that do produce high coverage navigation meshes (Hale, Youngblood, and Dixit 2008). The work here extends that research.

Motivation

Most commonly used spatial decomposition techniques focus on how to decompose the ground planes of game worlds. Worlds with multiple ground planes are generally decomposed one plane at a time, and special vertical transitions are added. This method of abstracting the 3D representation of a game world to a series of 2D planes is similar to how blueprints represent the floor plan of a multi-story building. These techniques generally assume that the agent's movement is restricted to the ground plane. The problem is that advances in level design and game play elements in games are producing increasingly complex and interactive levels. Game levels used to be designed such that in complex worlds such as cities, road levels would connect additional levels representing individual buildings. Flat image place holders for those building would be presented on the connecting road level, and users would initiate a load screen instead of a smooth transition into the building. Now, the buildings are integrated with the street areas and contain multiple walkable areas allowing seamless interactions through both doors and windows, all of which are transitions from one navigation mesh to another and require manual linking. These higher complexity levels will require more special case connections between different 2D decompositions, which makes generating 2D decompositions and then linking them by hand tedious.

We improve upon existing approaches for decompositions of 3D environments that does not require the world to be simplified into 2D planes and instead performs a true 3D decomposition, which subdivides the open space present in the world into a series of 3D regions. We provide a new method inspired by the 2D Adaptive Space Filling Volumes (ASFV) algorithm to allow spatial decompositions to operate on 3D geometry. Since we are drawing on ASFV for this algorithm, the positive features that ASFV decompositions contain, such as convex high-order polygons, high average minimum interior angles across all regions, good object containment, information compartmentalization, very high to perfect coverage of the level geometry, and a low number of total regions, are also present in our 3D decompositions. Using our new method, we can automatically decompose levels that include multiple ground planes and complex geometry. We accomplish this by transforming ASFV from a 2D algorithm that grows a series of quads into a 3D algorithm that grows a series of cubes. In addition, we altered the manner in which additional regions are added to the world via seeding in order to allow a more natural and usable fit to the affordances provided by the geometry.

Related Work

Most of the commonly used spatial decomposition techniques are limited to 2D representations of the world. A good overview of how these 2D techniques work can be found in (Hale, Youngblood, and Dixit 2008) or (Tozour 2004). In addition to these traditional 2D techniques, there are several algorithms that work natively in 3D.

Recently, work has been conducted to create 3D 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 can stand in. By connecting adjacent standable areas a walkability map can be generated. However, decompositions generated by this algorithm are limited to constant cell sizes, usually the size of the agent that navigates the world (so that the agent can stand in every cell), and no simplification is done on the resulting graph. This tends to produce meshes in which relatively small areas have a large number of regions.

Automatic Path Node Generation (Ratcliff 2008) is another 3D algorithm for navigation mesh generation. In this algorithm, the world is tessellated into a series of triangles. This list of triangles is culled down to places that a character in the game world could stand upon. At this point the algorithm finds the centroids of each triangle. These centeriods are transformed into rectangles by following simple space filling volume rules. These new rectangles are checked for collisions with world geometry and any invalid rectangles are discarded. Next, the algorithm attempts to calculate paths between these rectangles by trying to walk a character through the game geometry and seeing which rectangles are accessible to each other. This information is used to build the final connectivity graph, which creates a navigation mesh (or a series of connected disjoint meshes). This approach works well for agents that just walk from point A to B, but does not inherently handle cases where the agent can move via methods other than walking such as jumping or climbing.

Delaunay Triangulations can be directly extended into 3D to produce a purely triangular decomposition (de Berg et al. 1998). The Delaunay algorithm is straightforward—every vertex present in the world is connected to every other vertex to generate a series of triangular prisms such that they do not intersect any prisms already created. The algorithm then attempts to reform the triangles that compose these prisms in order to ensure that the average minimum interior angle of the resulting set of triangular prisms is maximized. This algorithm generates an excellent coverage decomposition that works well for navigation, but can create problem areas of small prism faces that cause problems with localizing objects to a single area.

Most waypoint based navigation methods can be extended into 3D, since they are just selecting points in space and finding the open paths between them. However, using such methods discards many of the benefits provided by having a navigation mesh. These benefits include such things as efficient information compartmentalization and the improvements to collision detection.

Methodology

The primary contribution of this paper is a novel algorithm inspired by ASFV to break up the free space in a 3D environment into a series of convex polyhedrons. This collection of polyhedrons will clearly define the empty negative space in the world in contrast to the positive space defined by the static world geometry (e.g., buldings). We refer to this algorithm as Adaptive Space Filling Volumes 3D (ASFV3D).

The Adaptive Space Filling Volumes (ASFV) algorithm, which inspired our 3D algorithm, operates in the following manner. First, the ASFV calls for the placement of a grid of unit sized quads into the world to be decomposed. These quads are provided an iterative chance to expand in every direction. At this point, the algorithm is very similar to the classic Space Filling Volumes. 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.

Adaptive Space Filling Volumes 3D

The Adaptive Space Filling Volumes 3D (ASFV3D) algorithm is described in Algorithm 1. There are two constraints on the input to ensure the validity of the decomposition. These constraints are similar to the ones for the 2D version of the algorithm and are as follows. First, it is assumed that all of the positive space regions in the input are convex. If the input regions are not natively convex they can be subdivided into convex regions. An important invariant which should be pointed out here is that our own generated regions end every phase of growth in a convex state. Second, once a free area has been claimed by a region, then that region must maintain its ownership of that area.

Our algorithm begins in a state we refer to as the initial seeding state by planting a grid based pattern of single unit regions across the environment to be decomposed. If the proposed location of a region is contained within a positive space area, it is discarded. This grid extends upward along the z-axis as well. After placement, the seeds are allowed to fall in the direction of gravity until they hit either the ground or an obstruction at which point they stop. If this falling results in multiple seeds landing in the same place then duplicates are removed. Our regions are initially spawned as cubes with 6 faces given in a clockwise order from the closest vertex to the origin point and then the top and bottom faces. These cubes are generated to be axis aligned. After being seeded into the world each region is iteratively provided the chance to grow. There are two possible cases for successful growth. The *base case* occurs when all positive space (impassable) regions are axis aligned. The more advanced growth case allows for non-axis aligned convex regions to be present among the positive space regions.

First, we shall examine the *base case* for growth in our algorithm. Each region is iteratively selected and provided the opportunity to grow once each frame. Growth occurs in the direction of the normal to each face of the region. We attempt to move the entire face in a single unit in this direction. We then take our proposed new shape and run three collision detection tests. We want to ensure that no points from our growing shape have intruded into a positive space or another region and that no points from either of the aforementioned obstructions would be contained within our newly expanded shape. Finally, the region performs a self check to ensure it is still convex in its new shape. Given that all these tests return acceptable results, we will allow the shape to finalize itself into that new configuration. If any of those conditions are unacceptable then it means that we had a collision. If the world is axis aligned when we reach this state we know that we were parallel to, as well as adjacent to, the shape we have collided with in our last size, which

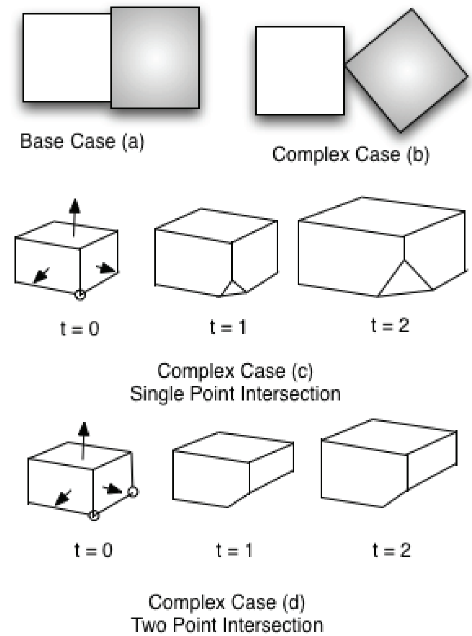

Figure 1: An illustration of the various cases present in ASFV3D. All growing negative space regions are shown in white. Growth is shown with an arrow. (a) Shows the basic growth case, (b) shows the complex case where growth is stopped by encountering a vertex of positive space, and (c) shows the complex case where the negative space region splits to adapt to a single vertex colliding with positive space. The colliding vertex is marked with a circle; the positive space is not shown to allow the algorithm's response to the collision to be visible. (d) Shows an example of the collision case where two vertices collide with a positive space obstruction.

is the desired ending condition for region growth as seen in Figure 1(a). In this case, we return to our previous size and set a flag to never attempt to grow that face again. We then allow every other face in the region to grow in the same manner. Once each face of a region has been provided the chance to grow a single unit, we proceed to the next region. This method of growth is sufficient to deal with all cases for axis aligned positive space regions.

The advanced case for growth in the algorithm deals with positive space regions that are not axis aligned. For the advanced cases, the algorithm begins by incorporating every step in the base case and expands on it. The algorithm cycles through each region and provides each face in the region with an opportunity for growth. Here, unlike the base case, we cannot use the properties of the axis aligned world to conclude that we are parallel to the object we have collided with. Hence, we will need to take some additional steps to ensure that we arrive at a decomposition with good coverage. In particular, we have to consider four separate cases of collision. Of these four cases, three cases are based on the number of the vertices in the growing face that have collided with a positive space area. The fourth case arises when

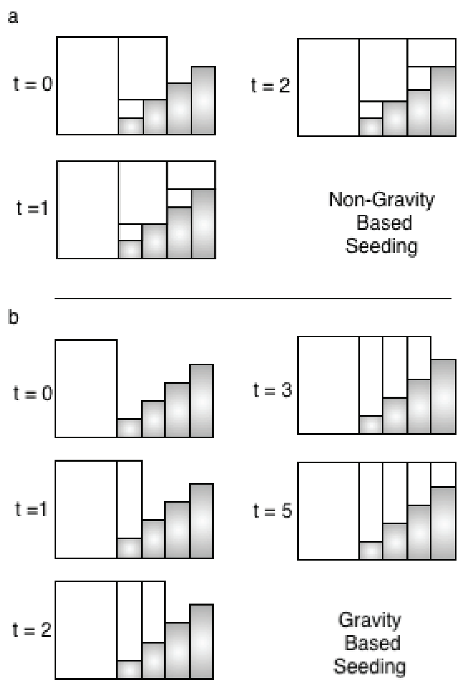

Figure 2: An illustration of ASFV3D seeding its way up a stair case. In each timestep a new seed is placed and then allowed to grow as much as possible. Positive space regions are shown in grey, negative space regions are shown in white. The world is viewed from the side and extends towards and away from the viewer. (a) Shows the results of seeding a world without applying a gravity model to place seeds. (b) Shows the results of seeding a world using gravity to modify the seeds locations. Shows the results of seeding a world without applying a gravity model to place seeds.

a vertex from a positive space object intersects a negative space region.

The first advanced collision case occurs when three or more vertices on the face of a region intersect a positive space in the same growth step. It follows that we have encountered a plane which is parallel to this growing negative space region. In this case, despite the fact the entire world is not axis aligned, the two faces we are currently considering are axis aligned and this can be addressed by the base collision case in which we simply stop growing.

The collision case resulting from encountering one or more *vertices* of a positive space object is actually the simplest of all collision cases. In this case, the face that intersected the object steps back to its last non-colliding location and further growth is ceased in that direction. It might seem strange that when a positive space region is encountered in this manner that the algorithm stops trying to decompose in that direction, but there is no way to achieve a better approximation of the colliding object without introducing concave regions or reducing our coverage as shown in Figure 1(b). The gaps in the decomposition resulting from this case will be filled via seeding and will be discussed later.

Algorithm 1: ASFV3D($NegativeSpaceRegions$)

```
StillGrowing = true ;
/* Populate the world with the initial
   user defined seeds */
if NegativeSpaceRegions.isEmpty() then
  seedWorld();
while StillGrowing do
  StillGrowing = false ;
  for NegativeSpaceRegion in World do
    for Face in NegativeSpaceRegion do
      Face.Translate(Face.Normal);
      if Face.isNotColliding() then
        StillGrowing = true ;
      else
        /* A collision has occurred.
           First check to see if it
           possible to increase the
           order of the polyhedron */
        if Face.isSplittable() then
          /* Adapt the face to
             follow the surface it
             intersected.  Insert an
             additional face at the
             point of collision
             which shares edges with
             every face that
             contained the vertices
             that are being split.
             */
          Face.SplitPoint();
          /* Lock the growth of the
             newly created vertices
             to lie on the equation
             of the plane they
             intersected */
          Face.ConstrainPoint();
          StillGrowing = true ;
        else
          /* The collision cannot be
             handled by splitting */
          Face.Translate(−1 *
          Face.Normal);
          Face.canGrow = false;
List seedPoints = new List ;
/* Find places to place new Seeds in the
   world */
seedPoints.Append(World.Seed());
if not seedPoints.isEmpty() then
  /* Restart growth algorithm on the new
     points */
  ASFV3D(seedPoints);
/* Run combining and clean up procedures
   */
World.combineConvexShapes();
World.removeColinearPoints();
World.RemoveDegenerateFaces();
```

The final two cases for collisions with world geometry involve inserting an additional face into the expanding region to closely adapt to the positive space geometry it encountered. The first case occurs when a single vertex from a growing negative space region intersects with a positive space obstruction. In this case, the vertex and each edge leading to it is split into a new face composed of three new vertices. The normal of this face is set to the negation of the normal of the positive space face it collided with and three points defined to lay directly on the positive space face. From this point forward the new points are restricted to only lie on the plane with which they have intersected. This means that when the other faces of the negative space regions grow they will pull this new face across the face it intersected. These new points are restricted from growing beyond the plane to prevent more non-axis aligned geometry from being exposed to the world. The results of this decomposition are shown in Figure 1(c).

The next case occurs when two points simultaneously intersect the same positive space face. In this case, a new face needs to be inserted into the negative space region in order to better approximate a positive space obstruction. Both of the intersecting points are split in this case resulting in four points that will form a new quad shaped face. It follows that if exactly two vertices intersect the same face of another shape then the entire edge between these points also intersects that shape. This means that we are, in effect, splitting that edge to become a new face. This new face is once again created using the negation of the normal of the face it intersected (as its normal) and made coplanar with the face it intersected. These new points are locked such that they can only move on the face that they intersected (for the same reason as in the previous case). This case is illustrated in Figure 1(d). This will allow near complete decomposition of the free space in close proximity of negative space without violating any of the underlying assumptions of the algorithm.

The growth techniques described above decompose the world reasonably well, but to assure a complete decomposition additional steps are required. As in traditional ASFV, we employ a seeding algorithm to decompose the free space that might have been initially missed. This procedure is outlined in Algorithm 2. Once all of the regions initially placed into the world have grown to their maximum extent the seeding algorithm is initialized. Each face of every region is given an opportunity to produce a seed in the world. The best approach for this seeding is to locate each distinct pocket of free space adjacent to a face and place a seed in it. It is extremely important for the quality of the decomposition that these seeds are allowed to fall according to the world gravity model, stopping only when they hit some positive space.

Applying gravity to seeding produces a much cleaner and more usable decomposition. Consider the two examples given in Figure 2, which shows possible methods of seeding a staircase from a negative space region at the bottom of the stairs. In the case shown in Figure 2(a), gravity is not applied to the seeding, and the initial seed grows out skipping over the first stair. Additional seeds are then placed above and below this first region until the entire stair case is decomposed. This creates a navigation meshes that implies that agents can float up from the bottom of the stairs and end up half way up the stair case, which most likely is not true. In the better decomposition shown in Figure 2(b), seeds are affected by gravity. In this case, a seed is generated from the first negative space region and then allowed to fall to the floor of the stair directly adjacent to it. This seed then grows to fill this single stair and all of the space above it. After growing, this new region will generate another seed that fills another stair completely. This cycle will continue until the staircase is completely decomposed. By comparing the two generated decompositions for the staircase it is obvious the decomposition with gravity generates a more usable decomposition as it is possible to stand in a single region on a stair. This is not possible for most of the stairs in the non-gravity based seeding algorithm, as many of the regions on the stairs do not properly represent how stairs are used.

Aside from the incorporation of gravity-based seeding, the seeding system in ASFV3D is identical to the 2D version and allows the algorithm to achieve complete decompositions of free space. This seeding system is especially effective in the case discussed above where an obstruction intersects the face of a free space region. After the seeding algorithm has concluded, the main growth algorithm is called again on the newly placed seeds providing them with an opportunity for growth. This cycle of seeding and growing continues until no new seeds are placed in the world at which point the world is fully decomposed and the algorithm terminates.

Algorithm 2: Seed()

```
List seedPoints = new List;
for NegativeSpaceRegion in World do
    for Face in NegativeSpaceRegion do
        List possibleSeeds =
        Face.GenerateSeeds();
        for seedPoint in possibleSeeds do
            while seedPoint.isInOpenSpace() do
                seedPoint.translate(GRAV_DIR);
            seedPoint.translate(-GRAV_DIR);
        seedPoints.extend(possibleSeeds);
seedPoints.removeDuplicates();
return seedPoints;
```

Experimentation

We evaluated the ASFV3D algorithm by comparing it with two 3D spatial decomposition algorithms: Extruded Space Filling Volumes (ESFV) and Automatic Path Node Generation (APNG). For our testing environment we wanted a game world feature that would be a stumbling block for most 2D based decomposition algorithms. Hence, we chose a staircase with a non-axis aligned ramp leading up to it as our test example. There are three main reasons for this. First, a set of stairs contains many walkable steps, each of which is set at a unique height above the ground, so even individually a staircase is actually difficult to decompose. Algorithms dependent on projecting each ground plane level into 2D must project each step into 2D separately which is time consuming, or the stair case decomposition will have to be performed by hand and linked into the different levels of decomposition it connects as a special case. Secondly, while there are multiple possible decompositions for this test case, some forms of decompositions are dramatically better than others for use in agent navigation as shown above (Figure 2). Third, due to the number of regions present in more complex decompositions, it is hard to visualize the decomposition, so we felt a simple but difficult test case would best illustrate the capabilities of ASFV3D. The three algorithms generated the decompositions seen in Figure 3.

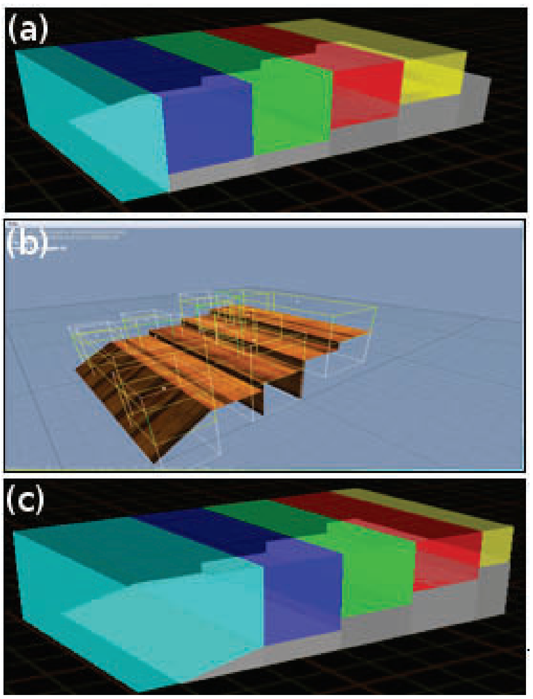

Figure 3: A comparison of multiple decomposition methods when building a navigation mesh for a stair case. (a) Extruded Space Filling Volumes. (b) Automatic Path Node Generation. (c) ASFV3D

Table 1: Comparison of Multiple Spatial Decomposition Algorithms

Algorithm	Number of Regions	Coverage
SFV	5	70%
PATH NODE	12	90%
ASFV3D	5	100%

The results of decomposition for each of the three algorithms were compared in terms of how many regions were produced and the coverage (i.e. the percentage of the empty space in the world contained in the decomposition). These results are summarized in Table 1. ASFV3D outperforms both ESFV and APNG in terms of coverage, and is the only decomposition algorithm to provide complete coverage of the world. Having a high coverage decomposition is important for tasks such as pathfinding, information compartmentalization, or collision detection. This is because, as the coverage percentage drops, gaps and unwalkable areas form in the navigation mesh, which dramatically reduces its usefulness. Results also indicate that ASFV3D is comparable with SFV when it comes to producing the fewest regions. This is an important consideration as fewer regions means a reduced search space for path finding algorithms or other graph search algorithms. Overall, when both coverage and number of regions are taken into account ASFV3D produces the best decomposition for the difficult test case presented here.

Conclusion

Adaptive Space Filling Volumes 3D provides a fresh take on decomposing space in 3D. Instead of using 2D simplifications and extrusions it operates in the 3D free space present in the world environment. This technique provides higher coverage and good decomposition even in difficult to decompose areas such as stairs or other vertical transitions. In addition, since this algorithm decomposes the entire world at once, it removes the need to join multiple different sections (e.g. floors) of decompositions by hand.

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.; 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)*.

Ratcliff, J. W. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.8 Automatic Path Node Generation for Arbitrary 3D Environments, 159–172.

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