Generating Texture-Aware Spatial Decompositions

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

Abstract

This work presents an algorithm to provide a better representation of space to artificially intelligent characters (i.e., agents or bots) in game and simulation environments by providing a more accurate breakdown of the traversable space present in the game environment. Such representations are generally constructed by decomposing the walkable space present in a game environment into a series of convex regions to form a data structure called a *navigation mesh*. We extend the basic concept of a navigation mesh by the introduction of an understanding of the textures that are attached to the underlying geometry creating what we refer to as a *texture-aware navigation mesh*. This does result in a more complex navigation mesh (more regions and a larger search space). However, since the textures of walkable geometry can be used to determine the appropriate traversal method for that terrain, a game character can determine valid paths for their traversal methods using just the navigation mesh (e.g., characters in cars can generate paths containing just roads or walking characters can create paths containing just sidewalks). We also present a use case that shows how such a system of texture aware navigation meshes might benefit character path planning and search in virtual environments. In this use case, we examine a Real Time Strategy game style game environment, which shows it is possible to generate a navigation mesh such that each region is composed of a single terrain type.

Introduction

Providing a high quality representation of the traversable (walkable) space present in game and simulation environments is one of the primary challenges when developing realistic artificially intelligent (AI) characters to operate in these environments (Tozour 2004). The base level geometry and other obstructions present in the environments are generally available to the character's AI, but this listing only tells the character where they cannot walk. It does not provide any sort of organized listing for the walkable areas. The walkable area in such an environment is presented to a character in one of two general ways. First, there are sparse representations of the environment. These representations generally consist of known good points in the environment and the known good paths between them (e.g., waypoints). The second commonly used type of representation is a dense spatial representation. In this style all or nearly all of the walkable space present in the environment is broken down into regions. These regions are generally convex—convexity is maintained so that a character can go from any two points within a region and be assured of remaining within the same region. These representations allow path planning and other searches through open space to be conducted using a graph or topological map search (the regions become vertices and adjacent regions are connected by edges). Using a set of convex regions to represent the walkable space in an environment creates a data structure referred to as a navigation mesh. In both dense and sparse representations characters search the representation to generate a path and use that path for global planning through the environment. In addition to global planning, characters utilize local path planning to deal with areas not contained in the representation or obstructions that are too small to be present in the navigation mesh.

In recent years navigation meshes have become the search space representation of choice in game and simulation applications (McAnils and Stewart 2008). Using a variety of algorithms (see Related Work section) we can generate high coverage navigation meshes by decomposing the walkable space present in the environment. Then character path planning becomes a matter of locating the start and end regions with a point in convex polygon algorithm and then executing a graph search algorithm (A*, D*, D* Lite, etc.) to find the shortest path between the start and end regions.

However, one potential flaw with existing navigation mesh generation algorithms is that they treat all traversable space as the same. This means that a navigation mesh for an urban environment centered on a single building with traversable roads, sidewalks, and grass would treat all of these different surfaces as the same in terms of walkability. Furthermore, regions of the navigation mesh might well cross over multiple different terrain types as long as they are all within the specified tolerance for differences in height for the spatial decomposition. This could result in problems for agents attempting to utilize this navigation mesh. For example, an agent driving a car though this hypothetical mesh would not know if they were on the road, the grass, or even the sidewalk based on the navigation mesh. Furthermore, paths generated using this navigation mesh might not

actually be traversable depending on how the agent is supposed to traverse the environment (e.g., the path might contain only grass or sidewalk regions, which are generally not traversable if the agent is driving a car). This creates additional overhead as the characters must still run some form of local path planning based on the terrain they are traversing. Additionally, this creates the need for an another check over generated paths to ensure that they are actually traversable via the manner the agent intends to utilize them.

We propose a new algorithm that extends popular navigation mesh generation techniques, which will allow them to generate texture aware navigation meshes. This extension ensures that each region in the navigation mesh is only composed of a single texture type. These subsets of regions exist in the navigation mesh in a manner that is easy to query and ties the different textures to a traverse methods (e.g., road textures can be associated with the *drive* traverse type). By allowing characters to specify a traverse method along with the start and end points of a path, we ensure that the generated paths are entirely traversable by the chosen traverse method. We accomplish this by iteratively toggling the walkablity of each of the different terrain textures of the environment and generating a new sub-decomposition that is then appended onto the final decomposition.

Related Work

Any of the following commonly used full coverage methods of generating a spatial decomposition would work as the basis for creating a texture aware spatial decomposition:

The Delaunay triangulation algorithm is a well known method for generating a series of triangles from an input set of points and edges. Using this algorithm every vertex present in the world is connected to every other vertex to generate a series of triangles such that they do not intersect any triangles already created (Delaunay 1934). The algorithm then attempts to reform the lines that compose these triangles in order to ensure that the average minimum interior angle of the resulting set of triangles is maximized. This algorithm generates an excellent coverage decomposition that works well for navigation; however, it often contains more regions than are strictly necessary as it only uses triangles.

Navigation mesh construction via the Hertel-Mehlhorn (Hertel and Mehlhorn 1983) algorithm is commonly used to generate a listing of convex walkable areas (Tozour 2002) and for a time was considered to be near optimal for this purpose. This algorithm works by connecting all of the vertices of the world geometry that border on the walkable areas into a series of triangles. This algorithm can also consume as input a listing of triangles generated via some other triangulation method. Triangles have the inherent property of always being convex, which means we have already generated our delineation of the walkable space at this point. However, the contribution of the Hertel-Mehlhorn algorithm is to optimize this listing by combining triangles into higher order polygons. The algorithm calls for the removal of an edge from a pair of adjacent triangles such that the resulting shape remains convex. The removal of lines is then repeated until the algorithm is unable to find any acceptable lines to remove at which point it terminates.

It is also possible to generate a spatial decomposition and a navigation mesh using a growth based approach (Hale 2011). In this method a quad or other base shape is placed into the environment. This shape is then allowed to grow and expand outward in the direction of the normal of each edge. When the growing quad encounters an object it either stops growing or is subdivided into a higher order polygon depending on the geometry present at the point of collision. Once the initial quad has ceased growing, additional quads are placed into the environment and then allowed to grow outward in the same manner as the initial quad. This continues until the entire environment has been decomposed. The advantage of this algorithm is that it will only generate axis-aligned edges between traversable regions. This makes it easier to do movement planning and path smoothing between regions.

Methodology

In typical game environments different types of terrain have different textures associated with them. These associations between terrain types and textures are generally consistent within any given environment. Using some prior knowledge provided by the level designer we can group these textures into clusters based on common usage (designer specified) and attach a traverse method to each texture group. When generating decompositions to assist with agent path planning it makes sense to utilize these groupings to encode the terrain information into the regions of the navigation mesh in the form of traversal tagging.

We accomplish this by extending existing navigation mesh algorithms to run multiple times altering what is and is not considered to be an obstructed or non-walkable area and appending the resulting regions to the same navigation mesh. We rely on the user to generate a list of which textures correspond to any given traversal methods. For instance, the asphalt, road, and highway textures might all designate areas where the agent can *drive* while the sidewalk and grass textures indicate areas the agent can *walk*.

To implement texture aware spatial decompositions we first insert fake obstructions that are projected up from all of the ground plane(s) in the environment. Each of these fake obstructions is tagged with the texture of the ground plane triangle they are extruded from. Then all of the fake obstructions associated with a user defined traversal method are toggled off to be traversable again. The navigation mesh generation algorithm of the implementers choice is then executed on the world in its current state (in our example images we use a growth-based algorithm). All of the regions of walkable space generated in this cycle of the decomposition algorithm are tagged with the traverse method that is defined by the current texture set.

After the decomposition algorithm finishes execution, another set of these fake obstructions are toggled off and the decomposition algorithm is executed another time. Again, regions that are generated in this second execution are tagged with the set of traverse textures that were just toggled to be walkable. This cycle repeats as shown in Algorithm 1

Algorithm 1: Texture Aware Spatial Decompositions

```
List TextureKeywords;
List GroundTriangles;
NavigationMesh CurrentMesh;
for Keyword in TextureKeywords do
    /* Determine which sections of the
       groundplane are traversable for
       this texture keyword */
    for Triangle in GroundTriangles do
        /* The matches method can
           utilize everything from
           simple string matching to a
           designer provided list of
           texture names */
        if Triangle.matches(Keyword) then
            /* Mark the triangle as being
               passable in this phase of
               the decomposition */
            Triangle.setWalkable();
        else
            /* Mark the triangle as being
               impassable in this phase
               of the decomposition */
            Triangle.setUnWalkable();
    /* Generate a new set of regions
       using just the current texture
       and add them to the existing
       navigation mesh */
    List tempRegions =
    Decompose(GroundTriangles, Keyword);
    CurrentMesh.append(tempRegions);
/* Only combine adjacent regions if
   the result would be convex and
   both components have the same
   traverse method */
CurrentMesh.specialCleanUp();
CurrentMesh.buildNavigationLinks();
```

until there are no further traversal methods to be considered and all of the traversable space in the environment has been decomposed into regions.

To utilize the navigation mesh generated by this process, gateways are defined to exist between regions of the same traverse types. Connections between regions which have different traverse types are referred to as boundaries and can have special traverse types (e.g., a bicyclist might be able traverse both the road and the sidewalk, but might have to *hop* to move between the two types). Such special traverse methods on boundaries must be provided by the end user and can serve to signal characters that special animations should be played when moving between two such regions.

There are two special considerations that must be dealt with when constructing a texture aware spatial decomposition. First, it is possible that there are certain terrains or texture types that might be present in one or more traverse types (e.g., the crosswalk would be present both in the *drive* and *walk* traverse types). This dual representation is accomplished by considering such textures to be distinct from parent texture traverse groups and adding additional dual use traverse groups (so there would be traverse groupings of *drive*, *walk*, and *drive or walk* in the case of the crosswalk). Like the original texture groupings and traverse classifications these special considerations must be provided by the end user. Secondly, the manner in which regions are combined when cleaning up a spatial decomposition has to be slightly altered. Normally, adjacent regions are combined if the resulting region would be convex. If we utilized this cleanup method in a texture aware decomposition we might combine regions associated with two distinct traverse types. Instead, we have to check that both regions share the same traverse type before we can combine them.

Finally, while we have discussed generating the groups of textures and associated traverse types using user provided lists, we found in our use cases that it was possible to automatically generate most of these associations. When game designers create environments they generally use descriptive titles for the textures associated with components of the environment. Using a simple string matching algorithm and some common words (e.g., “road”, “grass”, “concrete”) we were able to automatically determine which grouping most of our textures belonged in. In the future, we plan to expand this automatic grouping using some lexical databases such as ResearchCyc (http://research.cyc.com/) or ConceptNet (http://conceptnet5.media.mit.edu/) to improve this automatic classification.

Case Studies

It might seem that it would be best to compare the navigation meshes generated by texture aware spatial decompositions to the existing approaches used to generate representations that recognize the different traverse types present according to the terrain of the environment (e.g., multiple waypoint graphs or occupancy grids). However, this is not an informative comparison. The superior performance provided by navigation meshes when compared to sparse-spatial representations or the excessively complex occupancy grid representations will eclipse any gains or losses that occur when texture aware spatial decompositions are considered (Tozour 2002). By the same token it might also seem that comparing the navigation meshes generated by traditional spatial decomposition techniques to the ones generated by texture aware decompositions would be informative. However, the metrics on which navigation meshes are compared are designed to look at the number of regions and the coverage of the world. Even the advanced decomposition metrics presented in (Hale 2011) are focused primarily on optimizing path planning by minimized the size of the search space. When generating a texture aware decomposition we are intentionally producing more regions in the navigation mesh to better represent how the underlying terrain in the environment should be traversed. In short, texture aware decompositions expose additional features that compensate for the fact they contain more regions and as such a comparison with existing metrics would be invalid. Instead, we present a

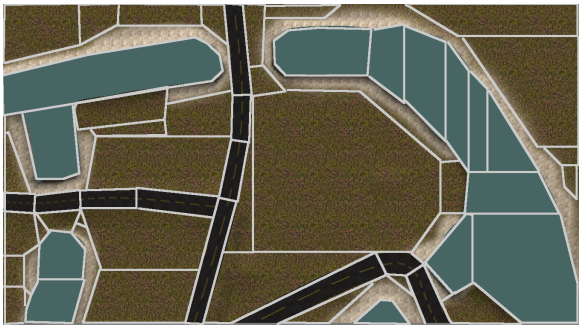

Figure 1: An exterior RTS environment showing a river and a road network, along with the muddy, hard to traverse ground between them.

case study showing how such a texture aware spatial decomposition might be used to better represent a game or simulation environment.

Our use case considers an outdoor environment composed of several different terrain types composed of roads and rivers crossing as shown in Figure 1. Additionally, this environment contains muddy fields that are impassable to wheeled vehicles and can only be traversed by tracked vehicles. In this environment the cliffs facing the water shown in white are not traversable. This is the type of environment that would be commonly seen in a Real Time Strategy (RTS) game (our level design is inspired by the sample levels used by (Julio Obelleiro and Cerpa 2008) in their work analyzing RTS levels). In a RTS game, character movement is less of a question of *should* a character be allowed to move through an area, and more a question of *can* this character move through the terrain present in the area given their movement methods. In the Dawn of War games there are characters that have to move around all obstructions, and other characters that can crash through some subset of obstructions. Additionally, it is quite common to see game characters that have other movement types that enable them to move through areas that would be impassable to other characters (i.e., flying characters that can cross water, or very tall characters that can step up or down cliff sides). The traditional solution to this problem of multiple traverse types attached to different characters has been to produce a different spatial representation for each type of traversal. However, this method can result in considerable storage overlap as the more general traversal types often overlap areas that could be traversed in some other manner (e.g., a representation of all the areas a character could *fly* through would overlap an area showing locations a character could *walk*).

With the introduction of a texture aware spatial decomposition and the resulting navigation mesh, we can properly quantify which traverse methods are applicable to each area of the environment. This allows us to discard the multiple potentially overlapping spatial representations that are traditionally used for such environments, and instead use a single navigation mesh. It is important to note that when implementing a navigation mesh in such a manner either the regions or the characters need to be aware of a hierarchy of traverse methods (i.e., there must be a way to communicate that regions which are tagged as *walk* are also acceptable for characters that use the *fly* traverse method either by including multiple tags per region, or allowing characters to employ multiple traverse methods when path planning and moving). This results in a general reduction in the number of data structures that must be maintained—one global structure to support multiple traverse types instead of one structure for each traverse method. Additionally, there is also (usually) a reduction in the size of the search space inside the navigation mesh by using the traverse method as a filter on potential regions of the navigation mesh when path planning.

Conclusion

In this paper, we have presented an algorithmic extension to existing spatial decomposition algorithms that allow for the generation of navigation meshes that are biased to account for the textures of the underlying geometry. This texture biasing allows us to represent each of the traverse methods present in the game or the simulation with a tagged subset of the regions in the navigation mesh such that each region is associated with a single texture or group of textures. These textures are then associated with a traverse method and agents are able to query the navigation mesh to only generate paths that utilize their available traverse methods. This produces character path plans that we know are traversable for that character as every region in the path is composed entirely of terrain that they can traverse. We then applied texture biased decompositions in a case study showing the applications of such decompositions in a RTS environment.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. OISE-0730065.

References

Delaunay, B. 1934. Sur la sphere vide. In *Classe des Sciences Mathematiques et Naturelle 7*.

Hale, D. H. 2011. *A growth-based approach to the automatic generation of navigation meshes*. Ph.D. Dissertation, Charlotte, North Carolina, USA.

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

Julio Obelleiro, R. S., and Cerpa, D. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 4.1 RTS Terrain Analysis An Image-Processing Approach, 361–372.

McAnils, C., and Stewart, J. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.4 Intrinsic Detail in Navigation Mesh Generation, 95 – 112.

Tozour, P. 2002. *AI Game Programming Wisdom*. Charles River Media. chapter 4.3 Building a Near Optimal Navigation, 171.

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