Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds

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

Abstract

This paper presents a new method for decomposing environments of complex geometry into a navigation mesh represented by bounding geometry and a connectivity graph for real-time agent usage in virtual worlds. This is accomplished by the generation of a well-defined and high-coverage set of convex navigable regions and the connected gateways between them. The focus of this paper is a new automated algorithm developed for decomposing a 2D representation of world-space into arbitrary sided high-order polygons. The DEACCON (Decomposition of Environments for the Creation of Convex-region Navigation-meshes) algorithm works by seeding a 2D polygonal representation of world-space with a series of quads. Each quad is then provided with the opportunity to grow to its maximum extent before encountering an obstruction. DEACCON implements an automatic subdividing system to convert quads into higher-order polygons 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 Hertel-Mehlhorn navigation mesh decomposition methods, DEACCON provides more complete coverage, controllable mesh sizes, and better overall algorithmic control to desired decomposition quality with an improvement in agent navigation speed due to better decompositions.

Introduction

When examining the usage of agents in virtual worlds, whether games or simulations, one of the first questions that must be answered is how to represent the areas of the world that agents can navigate. The open areas, called free or negative space, are not explicitly defined like the obstructed or positive space are defined by the world geometry. There are no nice lists defining the different sets of walkable regions like there are for the geometry obstructing the world. Obtaining such a listing of negative space requires using a spatial decomposition algorithm. While several such algorithms do exist there are no freely avaliable general purpose automated tools to perform the decomposition, as such the complex implementation problem remains even after an algorithm is selected.

In this paper, we present an enhanced Space Filling Volumes (Tozour 2004) algorithm to decompose virtual environments into convex regions of walkable space. We also present a tool that implements this approach. Having this decomposition provides us with several distinct advantages for agent navigation in virtual worlds. It is possible to build a navigation map from a decomposition that shows the connectivity between regions as well as the distances between the regions. Before decomposition, in order to plan a path, the agent would have to probe through world space looking for collisions with the geometry at every point on the potential path. After the walkable space has been decomposed, the path planning algorithm only has to consider the current region, the target region and the connectivity between the regions. Using this information any of the common shortest distance path planning algorithms could quickly generate a path from the navigation map.

Decompositions of the world provide compartments in which information can be stored. By localizing the objects agents need to reason about into the same regions being used for navigation, we reduce the set of objects the agents need to consider when they reason. Agents only need to reason in depth about the region they are currently located in and possibly the neighboring regions. Regions that are further away do not require the same level of indepth reasoning. This also extends to objects in these regions. In addition, since we require that all discrete regions be convex, we ensure that everything inside a region will be visible to any agent located in that region. This eliminates the problem where agents have to reason about objects that are located around corners.

Finally, we are able to greatly accelerate collision detection with the world geometry. Using geometric techniques it is possible to determine quickly if an object is located within any of the free space regions. If an object is located outside the free space, then it is in collision with the world geometry. This process can be enhanced even more by starting a breadth first search centered on the last known location of the object. This will take advantage of the fact that objects generally do not undergo dramatic shifts from frame to frame and tend to be located close to where they were in the last frame. By doing this, it is possible to quickly resolve collision detection for most objects.

The area of spatial decomposition for agent navigation has not been explored fully despite being utilized extensively by the interactive entertainment industry. At least one top tier studio was not, until recently, using any decomposition methods and was evaluating across individual spatial coordinates for pathing (Sturtevant 2007). While it might seem that work done in triangulation of complex geometry for the purpose of rendering would apply, in actuality the needs of agents are radically different from those of the rendering pipeline. Our algorithm takes the simple Space Filling Volume work and adds two novel improvements to it. First, it allows areas that did not fully decompose additional chances to do so via seeding from existing decomposed regions. Secondly, it allows our growing regions to dynamically adapt to complex sections of the geometry by subdividing the regions into higher order polygons dynamically. These two additions allow for a much more complete and accurate decomposition of the environment. Finally, our approach provides the user with considerable ability to tune the manner in which the decomposition occurs in order to optimize it for the agent architecture they intend to use and then select the best decomposition from the potential permatations produced by our tools. In order to perform decompositions we must first perform a simplification on the world. We need to represent the 3D environment as slices in 2D. This process works much the same way that architectural blueprints can represent a complex 3D building by dividing it into floors. We do this by finding ground cutting planes that can be created from world geometry. Just as there are special custom generated regions added to blueprints to show the transitions between floors our approach will also require special markups to show transition areas between different levels of the decomposition. This abstraction works because, just as architects are concerned with the floor plan of each level when they create a blueprint, we are only really interested in the walkable space.

Related Work

While the research into the decomposition of virtual worlds is limited there has been some prior work done and a few techniques have been developed to simplify the task.

Space Filling Volumes (Tozour 2004) is a simple technique for generating a navigation map and forms the basis of our own algorithm. It works by placing square seeds throughout the environment and growing them outwards in all directions. In the case of a collision with the world geometry all growth in that direction stops. As a post processing enhancement generated regions can be combined if the resulting shape would still be convex. This helps to simplify the resulting navigation map. This technique works well for worlds where all of the geometry is axis aligned, but fails on worlds with arbitary or complex geometry.

Navigation Mesh construction via the Hertel-Mehlhorn (Hertel and Mehlhorn 1983) algorithm will also generate a grouping of walkable areas. This process works by connecting all of the vertices of the world geometry around the walkable areas into a series of triangles. Triangles have the property of always being convex. The algorithm then 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. Unfortunately, this algorithm causes certain problems for information compartmentilization. The corners of the world geometry are almost always filled with slivers of thin triangles (even after combining). Since these triangles are thin they are generally smaller than the objects you are trying to place into them. This means that objects will span across multiple regions and prevents agents from fully taking advantage of information compartmentalization with decompositions created from this algorithm.

By-hand decomposition of virtual worlds is still used by many areas of industry. Such hand decompositions are generally done following a heuristic to determine precisely how the decomposition should occur such as Hertel-Mehlhorn. These hand decompositions do provide excellent representations of world space and have both high coverage and good navigation potential. However, this method's biggest drawback is the extreme time requirement (several days per environment) to properly construct a hand decomposition and as such a better method is needed.

Finally, other methods of generating navigation meshes do exist such as Probablistic Roadmaps, Voroni Graphs, Waypoints, and others (Russel and Norvig 2003). These focus entirely on generating the paths for agents to travel and provide little to no information compartmentalization nor do they assist in collision detection.

Methodology

Our approach is based off of a simple physical event. The regions we place in the world resemble marshmallows that have been placed in the microwave. They expand dramatically to fill available negative space and then follow the contours of any positive space they encounter. This form of expansion allows us to establish very high degrees of coverage even in complex worlds.

The DEACCON (Decomposition of Environments for the Creation of Convex-region Navigation-meshes) algorithm is designed to provide the negative space breakdown to the other applications in the Common Games Understanding and Learning (CGUL) toolkit (playground.uncc.edu/GameIntelligenceGroup/Projects/CGUL). One of the other applications in this toolkit, SARGE, is responsible for generating the connectivity information between the positive and negative space regions. These two applications (DEACCON and SARGE) work together to produce Static Spatial Perception Service (SSPS)(Youngblood et al. 2006) data from the raw geometry input. Our work is designed to integrate strongly with SSPS. SSPS provides agents with information that is specially designed to represent 2D or 3D environments. It is a mapping of the positive (obstructions) and negative (walkable) space regions that exist in the world. It also provides connectivity and visibility information about these spaces, and an opportunity to store ancillary information for each region that might prove useful to agent-based AI in reasoning about place.

DEACCON, as shown in Algorithm 1 can be broken down into several simple steps. We have several assumptions and invariants we must maintain in order for our algorithm to be effective. First, we assume that all of the positive space regions provided as input are convex. Our own generated regions must end every phase of growth in a convex state. Finally, 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 that 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. Our regions are initially spawned as squares with 4 sides given in a counterclockwise order from the northwestern point. These squares 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 simple case occurs when all positive space (impassable) regions are convex and axis aligned. The more advanced growth case allows for non-axis aligned convex regions to be present among the positive space regions. Worlds which violate our assumption that all of the positive space input regions be convex are not handled by our algorithm at this time and will not be evaluated. However, by replacing non-convex geometry with overlapping sections of convex shapes it is possible to convert any world into something our algorithm can process.

First, we shall examine the *base case* for growth in our algorithm. Each region is selected and provided the opportunity to grow once each frame. Growth occurs in the direction of the normal to each edge in the region. We attempt to move the entire edge 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 another 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 growth. Given that all those 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. Because of the axis aligned properties in this state we know that we were parallel to, as well as adjacent to, the shape we have collided with in our prior condition, which is the desired ending condition for region growth. In this case we return to our previous shape and set a flag to never attempt to grow this region in that direction again. We then allow every other edge in the shape to grow in the same manner. Once each edge in a shape has been provided the chance to grow a single unit, we proceed to the next shape. This method of growth is sufficient to deal with all cases for axis aligned positive space regions.

The advanced case algorithm, as shown in Algorithm 1, is more complicated, but it is also able to deal with a greater variety of potential positive space regions. It begins by incorporating everything contained in the base case algorithm and then expanding on it. Again we cycle through each region and provide each edge in that region a chance to grow. This time, however, since we cannot assume that we will automatically be parallel to what we have collided with we need to take an additional step and ensure that we follow the contour of the region we have collided with. We have three basic collision cases to consider.

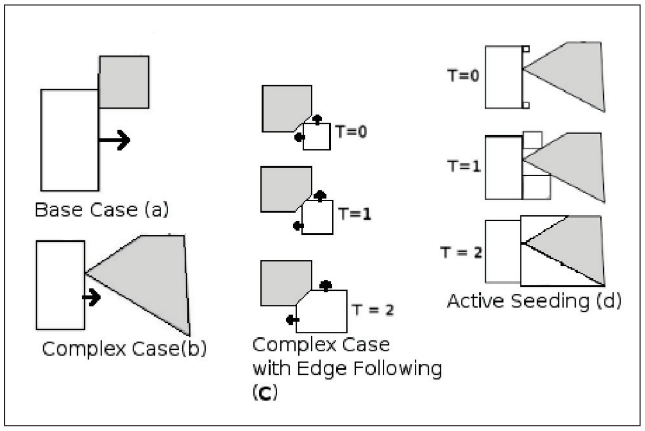

Figure 1: An illustration of the various cases present in DEACCON. All growing negative space regions are shown in white. Primary direction of growth is shown with an arrow. (a) shows the basic growth case. (b) shows the complex case where growth is stopped by encountering an edge. (c) shows the complex case where the negative space region enters contour following mode. (d) shows an example of seeding to generate new negative space regions.

The first is the basic parallel line case as shown in Figure 1(a). We can test this by comparing the equations of the edge we are advancing and the edge with which we have collided. If we enter this case, we proceed in exactly the same manner as in the case presented above since we have, in effect, shown that locally we are in the above *base case*. The next case occurs when a point from the object we are colliding with would lie within the newly proposed bounds of our shape. In this case we must stop further growth in this direction in order to preserve the convex property. In the case of this collision we are unfortunately forced to accept a poor proximity to the edges of the positive region as shown in Figure 1(b). We deal with this later by seeding.

The final collision case is the most interesting and occurs when one of the endpoints of the region edge would lie within a positive space region. We are able to split the vertex of our region into two points and follow along the contour of the obstacle, expanding the region and the order of the polygon in the process. We accomplish this by inserting a new edge into our polygon of length zero and connecting it to the two endpoints that suffered a collision. We then alter the direction of growth of each of the points that collided such that it is following the equation of the line with which it collided. We cap growth at the extent of the edge we are following such that we do not create additional non-axis aligned exposed edges to deal with later, which would be the case if we allowed modified edge growth to proceed past the edge it is following. Once we have overwritten the direction of growth for the contact points and limited their extent, we can return to following the *base case* and the region will grow to follow the obstructions as shown in Figure 1(c).

By following this method we are able to fit a region to non axis-aligned shapes such that the result will be axis aligned and allow the other regions to generate a better fit. Once every region reports that it is unable to continue growth we can proceed to the next step of the algorithm.

The above growth methods alone do not ensure a complete coverage of potential free space, but we employ a second algorithm as shown in Algorithm 2 to improve our results. We call the second algorithm *Seeding* and it works by locating areas adjacent to our regions that have not been covered yet as shown in Figure 1(d). We determine where to seed on an edge by looking at the areas of it that are in contact with obstructions. Once we locate every obstruction in contact with a given edge we can find each section of free space adjacent to that edge. A seed region is then placed into the midpoint of each free space area. This will result in a high degree of region coverage for the world. Once every edge has had a chance to seed we will re-enter the growth phases if there were any seeds generated. This allows our new regions the chance to fill any empty space and improve our decomposition. This is especially effective in collision cases where we were forced to stop growth due to collisions such as that in Figure 1(b). We continue to perform this cycle of grow and seed until we have filled in all reachable negative space.

Finally, we perform a cleanup and combining algorithm on our set of regions. We first go through and check for any regions which can be merged into one single convex region. After merging all allowable regions we can then go through and remove any degenerate zero length edges or colinear points to provide as clean an output as possible.

Experimentation

The DEACCON algorithm was tested and evaluated using five maps from a popular Quake 3 modification, Urban Terror (www.urbanterror.net). These maps were randomly chosen to cover a wide variety of level types, ranging from one map that is the interior of a building to another map that shows wide open spaces with general building geometry.

In the interest of space, we will be discussing our methods in reference to the level called Lakes seen in Figure 2 (A/B - Second from Left). We performed a series of decompositions on this level using three different decomposition methods. The first decomposition was performed with the Hertel-Mehlhorn (HM) algorithm (Hertel and Mehlhorn 1983). Figure 2 (C) shows the final HM decomposition. This approach to world decomposition in interactive domains is still very prevalent as many complex environments are decomposed by-hand or through only semi-automated methods. This method produced 42 regions after combining on the 2D ground plane.

Next, the fully automated technique of Space-Filling Volumes (SFV) as discussed previously was utilized to decompose the same world. The best uniform seeding to allow complete coverage was used, however it did experience connectivity and coverage issues. These issues are inherent to the SFV algorithm and not caused by our implementation. The decomposition is shown in Figure 2 (D). This method

Algorithm 1 DEACCON ALGORITHM

```
void startDEACCON(List NegativeSpaceRegions)
$StillGrowing = true$
// Perform Initial Seeding to populate world with
// regions based on user settings
**if** NegativeSpaceRegions.isEmpty() **then**
  seedWorld()
**end if**
**while** $StillGrowing$ **do**
  $StillGrowing = false$
  **for all** NegativeSpaceRegions in World **do**
    **for all** Edges in NegativeSpaceRegion **do**
      // Get a new Edge one unit foward in the direction
      // of the old edge's $Normal$
      $newEdge$ = currentEdge.Advance()
      **if** $newEdge$.isNotColliding() **then**
        $StillGrowing$ = true
        $currentEdge$ = $newEdge$
      **else**
        // We are adjacent to a positive space region
        // Check to see if we are in advanced case
        **if** $usingContourFollowing$ **then**
          **if** $newEdge$.isSplitableCollision() **then**
            // Adapt to follow Edge
            // Determine which Vertex collided
            // Add in extra point and edge
            $newEdge$.splitPoint()
            // Overide direction of growth to the
            // equation of the edge of the object
            // it collided with
            $newEdge$.setGrowthDirection()
          **else**
            // Not possible to grow in this direction
            $currentEdge.canGrow$ = false
          **end if**
        **end if**
      **end if**
    **end for**
  **end for**
**end while**
// List of points that will have seeds placed in them
List seedPoints = new List()
**for all** Regions in World **do**
  // Run Find Adjacent Open Space algorithm
  seedPoints.append(Region.findOpenSpace())
**end for**
**if** seedPoints.isNotEmpty() **then**
  // Re-start growth algorithm
  World.startDEACCON(seedPoints)
**end if**
// Run combining and cleanup algorithms
World.combineConvexShapes()
World.removeColinearPoints()
World.removeDeginerateEdges()
```

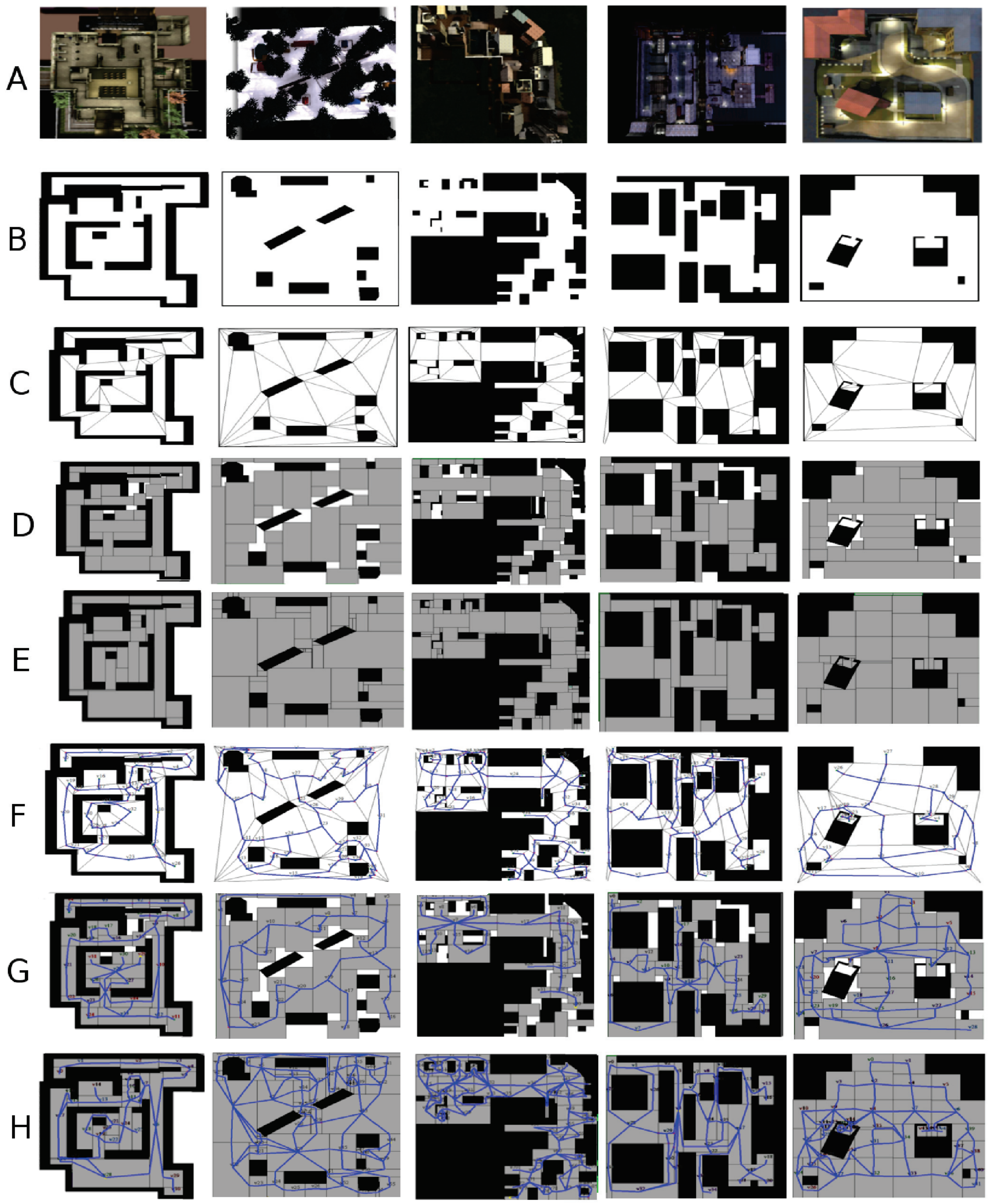

Figure 2: Images of the basic worlds (A) with each column representing a separate and unique world, the geometry we decomposed (B), the Hertel-Mehlhorn decomposition (C), Space Filling Volumes (D), DEACCON decompositions (E), and finally the last 3 rows show the connected navigation graph generated by each decomposition algorithm (F is Hertel-Mehlhorn, G is Space Filling Volumes, and H is DEACCON).

Algorithm 2 Locate points to add as seeds in open space

```
List findOpenSpace()
// Locate possible seeding locations
for all Edges of Region do
    // Determine all objects that intersect this edge
    // Determine midpoints of all free areas between
    // intersections. Compose a list of these points
    return Edge.getOpenAdjacent()
end for
```

produced 28 regions on the 2D ground plane after a simplification algorithm was run that combined regions where possible.

Finally, DEACCON was used to decompose the same environment. Figure 2 (E) shows the final decomposition. This method produced 73 regions after combining on the 2D ground plane.

After the negative space was decomposed, an analysis program was run that builds the navigation map between the centers of each region and the midpoints of common gateways (shared edges between regions). The navigation map for DEACCON and the other decomposition methods for the Lake map are presented in Figure 2 (F-H).

Table 1: Comparative Agent Performance on Decompositions Across Multiple Levels. * indicates statistical signficance with P-value of less than .05

Algorithm	Avg Distance	Avg Turns	Coverage
DEACCON	442.7*	4.12*	100%
SFV	505.3	3.97*	90%
HM	497.4	5.2	100%

These 15 decompositions were used for agent navigation. A simple agent using A* search (Hart, Nilsson, and Raphael 1968) was used to plan a path from the start location to the goal location using the centroids of the connecting gateways between regions and the centroids of the regions for navigation. This form of navigation map construction provides a base line for agent path planning. The actual path will be constructed from this navigation map using smoothing algorithms to provide a more natural looking path. An equivalance table was constructed to determine which map regions correspond to each other between different decomposition methods. Regions are considered to be equivalent to each other if their centers and extents are roughly the same. Eight paths between randomly selected regions were then created and the distances to travel those paths were calculated using the A* agent. This provides us with a total of 40 unique paths over each of the 3 decompositions. These results are then compared across all maps using a repeated measures F-Test design over the three different types of decomposition which are shown in Table 1.

The results show that there is a statistically significant reduction (pValue $<$ .05) in overall path distances using DEACCON. This is due to the less angular shapes that are produced over the more triangular decompositions of HM and the non-uniformity in size allowing for larger area coverage over SFV and HM. The larger areas also serve better in the localization of the agent and provide for better containment of objects (with fewer objects lying across region boundaries) to allow for segmentation and reasoning over regions or groups of regions more easily. The number of turns an agent has to make to complete each path was also tracked and evaluated. The paths generated by DEACCON contained statistically significantly (pValue $<$ .05) fewer turns than HM. These turns can cause delay in agent navigation and require additional path planning and consideration in order to compute a natural looking path even with smoothing algorithms. While DEACCON did not have fewer turns than SFV, in general SFV is a poor choice with which to build navigation maps. In this example SFV was given a larger than normal advantage in that all chosen paths were fully accessable to it. Normally, there will be large areas of the map that cannot be reached using SFV. Overall, speed performance for world traversal by an agent was improved over HM and SFV due to the shorter paths and fewer turns.

Conclusion

In conclusion the DEACCON algorithm provides both a fast and effective alternative to other methods of 2D geometric decomposition and navigation mesh construction. Agents have been shown to be able to effectively utilize the navigation paths generated using DEACCON and to accomplish tasks faster on average than with the other methods presented in this paper. In comparison to the only other algorithm to generate a complete coverage decomposition, Hertel-Mehlhorn, DEACCON was considerably quicker to construct intially and would be able to rapidly accept changes to the world geometry. DEACCON also provides the user with considerable control over the manner in which the decomposition occurs and how that final decomposition looks, which the other algorithms lack.

References

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4* (2):100–107.

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

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

Sturtevant, N. R. 2007. Memory-efficient abstractions for pathfinding. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)* (6):31.

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

Youngblood, G. M.; Nolen, B.; Ross, M.; and Holder, L. 2006. Building Test Beds for AI with the Q3 Mod Base. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.