Pathfinding in Open Terrain

S. D. Goodwin, S. Menon[†], R. G. Price

Centre for Interactive Digital Entertainment Research
School of Computer Science, University of Windsor
Windsor, Ontario, Canada
sgoodwin@uwindsor.ca, sam@iiita.ac.in, price9@uwindsor.ca

Abstract

Pathfinding involves solving a planning problem with agents seeking optimal paths from a start state to a goal state. The pathfinding process involves utilizing the full state space information available to agents to find the least expensive route to the goal. However most solutions to the pathfinding problem have solely focused on using graph based terrain representations and graph search algorithms to obtain a path. This paper replaces part of the search with a direct geometrical solution. We provide some preliminary indication of the potential merit of the approach.

1. Introduction

Pathfinding is a fundamental problem that typical commercial games must deal with in one form or another. It has been defined as the problem of finding a path linking two vertices of a graph [NAH04]. A common solution is to employ the A* search algorithm [HNR68]. The properties of this algorithm are well known. For example, A* is *optimally efficient*, i.e., for a given heuristic, no other optimal algorithm can guarantee expanding fewer nodes than A* [RN03; citing DP85]. Yet, in practice, pathfinding with A* can impose a severe resource impact in terms of memory and time requirements. This is especially true when multiple simultaneous pathfinding requests must be satisfied.

The literature abounds with approaches for overcoming the limitations of A*. Common themes include changing the search space representation (e.g., grids, waypoints, navigation meshes, etc.), changing the heuristic (e.g., Manhattan distance, Euclidean Distance, Vancouver Distance [Yap02]), pre-computing and/or caching paths, and sacrificing optimality to gain on speed by abstracting the search space (e.g., hierarchical approaches [BMS04]).

Many approaches to the pathfinding problem model the underlying world as a grid. The squares in the grid are then represented as the nodes of a search graph. Performance improvements are then sought using graph-theoretic algorithms and data structures. It is important to keep in mind that the graph representation is not a faithful model of the world; it fails to capture important spatial information (e.g., spatial proximity is not explicitly represented---only connectedness). An optimal solution in the graph is only an approximation of the optimal solution in the underlying world geometry. Path accuracy is affected by the size of the grid squares and the limited connectedness (typically 4 or 8 directions).

The lack of geo-awareness of such representations becomes most obvious in open terrain environments. In contrast to maze-like environments where A*'s expanding search fringe is constrained by walls and obstructions, in open terrains, expansion is only constrained by the heuristic and terrain cost. To find a path of a given length in the open is typically much more expensive than finding a path of the same length in a constrained environment. Approaches which are feasible for the confines of a dungeon fall down in the light of day.

This paper presents a preliminary proposal for an approach to pathfinding in open terrains that seeks to improve existing approaches by exploiting geometric information derived from the world model. While the approach is aimed at open terrain environments, it is equally applicable (without modification) to the deepest, darkest dungeons as well.

2. Pathfinding in practice

A* search is among the most popular pathfinding techniques adopted by commercial game developers. In conjunction with A* search, many different search space representations (each having advantages and disadvantages) have been tried. A useful introduction

[†] Current address: Faculty of Information Technology, Indian Institute of Information Technology, Allahabad.

to typical search space representations is given by Tozour [Toz04].

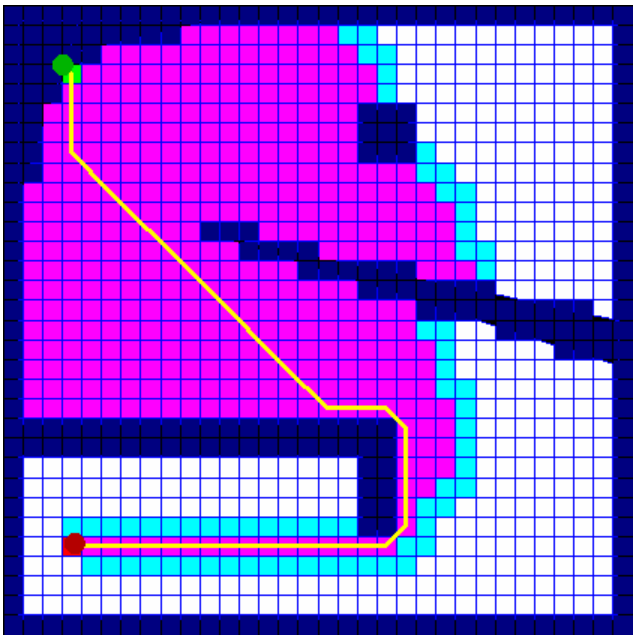

Figure 1: Grid representation [Toz05]

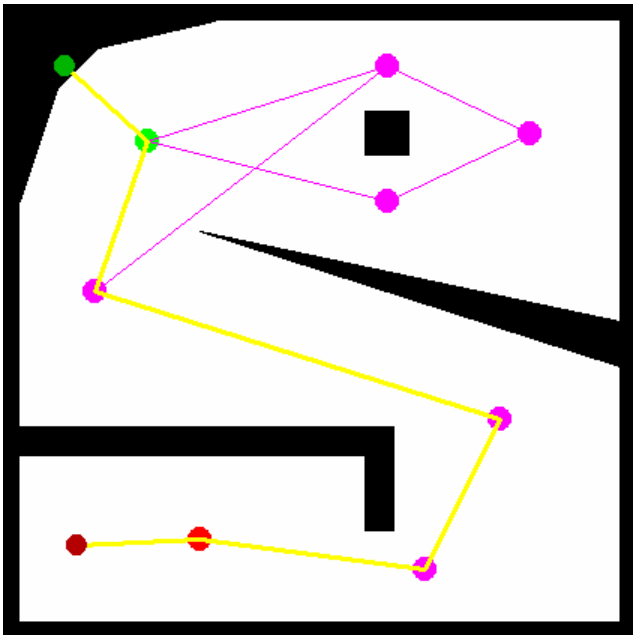

Figure 2: Waypoint representation [Toz05]

As Tozour explains, the simplest representation of the game world is a regular grid of squares, hexes, or triangles. Grids are most useful in 2D worlds. They support random-access lookup, that is, given the coordinates of a point in the world, one can directly determine the cell that contains it. However, since the size of the grid's cells impacts the quality of the solution path, grids typically require a large number of cells to adequately represent the underlying world (hence, a large search space). Techniques such as 'string-pulling' [Sno00] or spline fitting [Rab00] can be used to provide more natural looking paths.

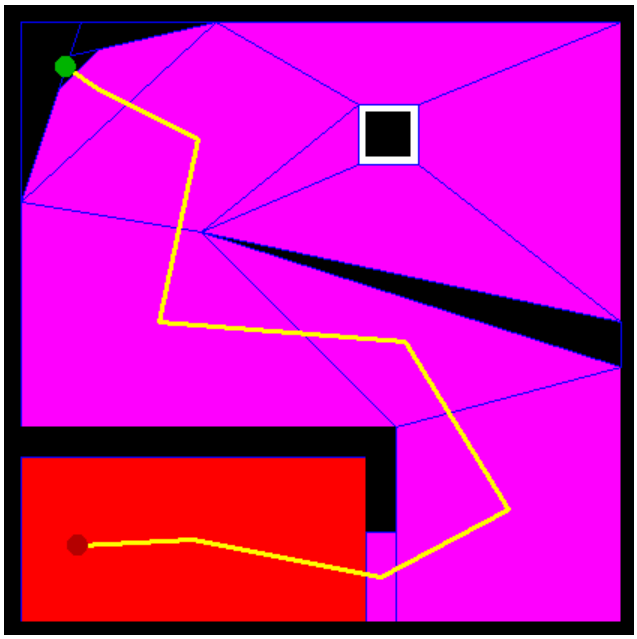

Figure 3: Navigation mesh representation [Toz05]

The advantages and disadvantages of representations such as waypoint graphs and navigation meshes are also covered by Tozour. The key issue is the trade-off between the size of the search space and the quality of the path found.

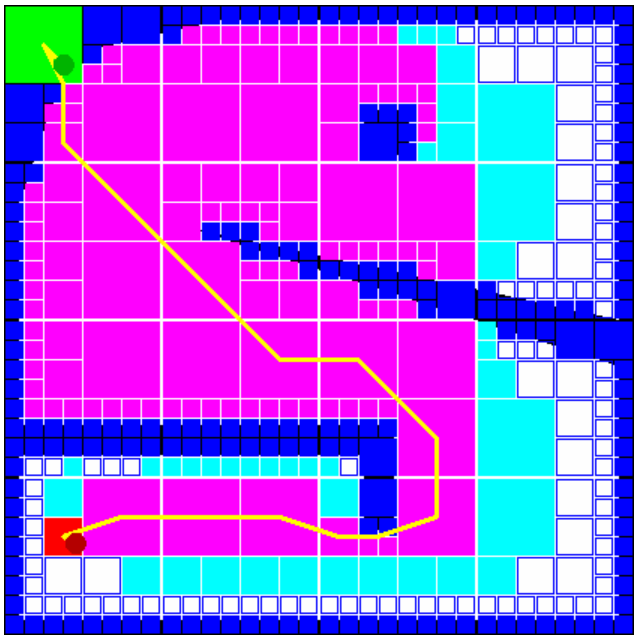

Figure 4: Quadtree representation [Toz05]

In unstructured open terrains, this trade-off is acute. One approach to this problem is the use of a quad-

tree representation. The map is subdivided into 4 squares. Each square that contains an obstacle is subdivided again. The process is repeated recursively until the area of a square reaches a specified minimum. The advantage of this approach is that large unobstructed areas are represented as a single node. The disadvantage is that the quality of the path is reduced.

Compare the nodes expanded versus path quality in Figures 1 to 4. The start and goal points are green and red, respectively. The start and end graph nodes are also green and red. The explored nodes (closed list) are indicated in purple and the fringe nodes (open list) are in cyan.

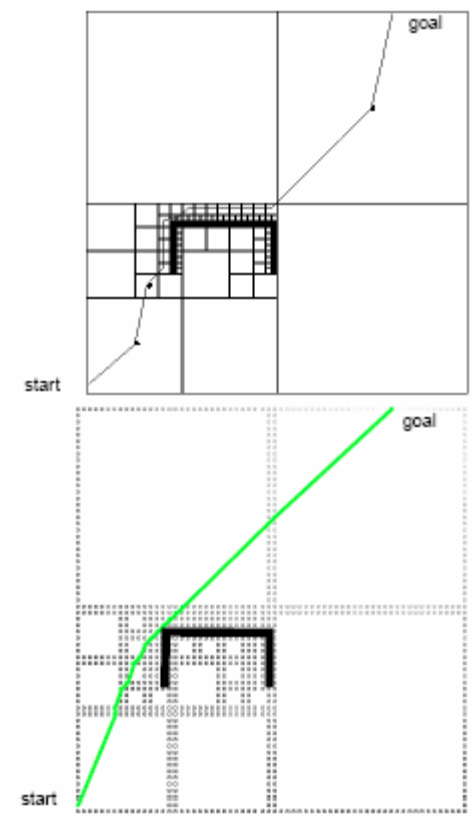

Figure 5: Framed quadtrees [YSSB98].

Note that in the quadtree approach, the path is constructed from quad center to quad center. For large quads, this can dramatically affect the path quality. To overcome this limitation yet attempt to retain, to a degree, the reduced search space advantage, another approach called framed quadtrees has been proposed [CSU97]. In this approach, high resolution cells adjoin the boundaries of the quads. Paths passing through a quad travel from boundary to boundary, crossing at a high resolution cell rather than going center to centre. Compare the path quality of framed quadtrees versus quadtrees in the illustration from Yahja et al. [YSSB98] in Figure 5. The improved path quality comes at the price of larger search space for cluttered maps.

One further approach is to partition the terrain into convex polygons covering regions of uniform terrain. Stout discusses several ways of doing this [Sto00]. An advantage of this representation is that because the polygons are convex and uniform cost, the locally minimal path traversing the covered region is a straight line. Since this line is guaranteed to remain in the confines of the polygon, its cost is simply its Euclidean length times the region cost.

In all the representations discussed here, the ultimate result is a graph to be searched using the A* algorithm which relies on a given cost function and heuristic cost estimator. One possibility for improvement is to provide a means to take advantage of geometric properties of the underlying world directly, rather than indirectly through search.

3. Our approach

As a starting point, consider a representation that is a hybrid of framed quadtrees and the convex polygon approach. Instead of treating the regions as nodes in the search graph, let the polygon edges be divided into a number of points based on the desired resolution. Take each of these points as nodes in the search graph. Nodes are connected if they correspond to points in the same polygon (points on the boundary between two polygons are each represented by a single node that belong to both polygons). The cost of the edges is now simply the Euclidean distance between them times the cost of the region covered by their polygon. Clearly, the properties of search using this representation are similar to that of framed quadtrees but with some additional flexibility from using convex polygons rather than quads.

Such an approach would be useful for open terrain environments except that, as the resolution increases, the search cost increases dramatically. To seek an improvement to this, consider the fact that the optimal path in a given search graph passes through the boundaries of a sequence of polygons. The path crosses each boundary at a point (or possibly through multiple points on a boundary). To determine which boundary points to cross the f-score for each corresponding node is calculated and nodes are selected in the usual A* fashion.

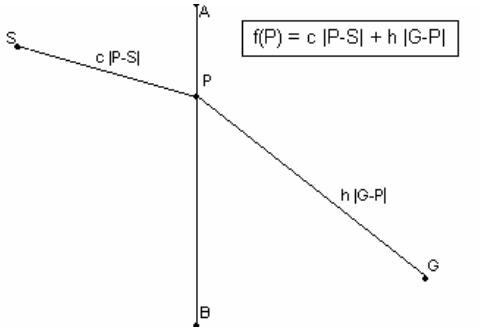

Figure 6: Find optimal boundary point problem.

In Figure 6, imagine S is the starting point which lies in a region with a boundary AB which is divided into points $\{A,P_1,...,P,\ldots, P_n,B\}$. Let G be the goal point in some possibly remote region. Let c be the terrain cost in the region containing S and let h be the heuristic cost factor (typically, h=1). For each, point on the boundary (and points on the other boundaries of the region containing S), A* must calculate the f-score of the point in order to later choose the next node to expand. For a point P on AB, f(P) = g(P)+h(P) = c|P-S|+h|G-P|.

Rather than determine the f-score of each point and thereby indirectly determine the point P on AB that minimizes the f-score, it is possible to determine P directly by an application of Snell's Law of Refraction of Light (we independently rediscovered this idea originally put forth by Mitchell and Papadimitriou [MP91]). According to Snell's Law, the path of a light ray passing through a boundary AB between regions with indices of refraction c and h satisfies the equation $c\cdot\sin\theta = h\cdot\sin\phi$ where θ and ϕ are the angles of incidence and refraction respectively. Snell's Law is derived from Fermat's Principle of Least Time (light follows the path of least time). In [MP91], an algorithm using a continuous Dijkstra technique simulates the effect of a wavefront emanating from S.

Our approach differs in that we integrate a boundary point solver (based on Snell's Law) into the A* algorithm. The idea is to dramatically reduce the search space by eliminating the need to evaluate individual boundary points while retaining (and even improving) path quality. To do this, we need to solve the problem of determining the boundary points on each of the boundaries a path cross between S and G by minimizing a heuristic cost function of the form $f(P_1,\ldots,P_k) = \sum c_{k-1}|P_k-P_{k-1}| + h|G-P_k|$ where P_0=S. Figure7 illustrates this for k=2 (the blue path).

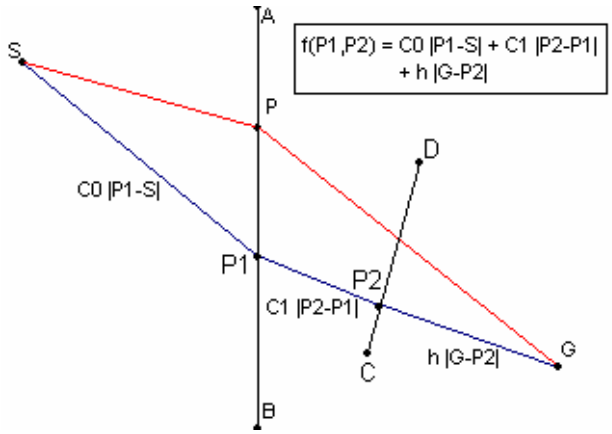

Figure 7: Finding optimal boundary points

4. BPSolver

In mathematical terms, minimizing the multivariable cost function produces in a system consisting of k quartic equations in k unknowns. Mitchell and Papadimitriou [MP91] have noted that elimination among these equations yields a polynomial of degree doubly exponential in k. Instead of seeking an exact solution, we adopt a commonly used optimization method known as iterative coordinate decent which works by iteratively updating the points to minimize the cost function.

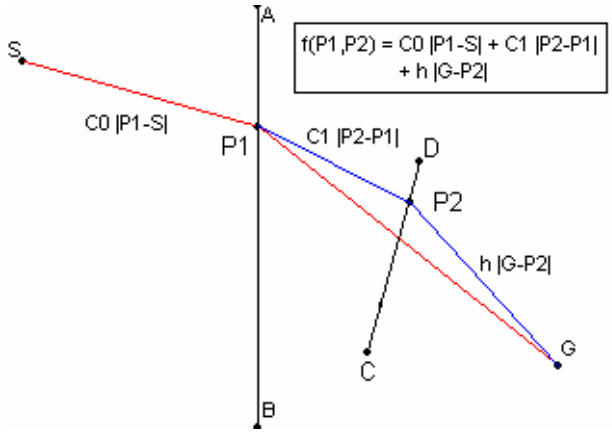

Figure 8: BPComputeOne function

At the heart of this iterative procedure (which we call BPSolver), is a function, BPComputeOne, that computes the exact solution for the locally optimal boundary point P given the prior and next point and prior and next region costs. In Figure 8, given P1 and G, BPComputeOne finds P_2 on CD which minimizes $f(P_1,P_2)$. It does this by finding the roots of the quartic equation that results from differentiating f with respect to P_2 (expressed in parametric form $P_2(t) = C + t(D-C)$ for $0 \le t \le 1$) and setting the derivative

equal to zero. Using Eberly's root solver [Ebe05], the cost function is evaluated at roots of this equation (or the closest endpoint if the root is out of range) to choose the minimum. The quartic equation is of the form $w_4rt^4+w_3t^3+w_2t^2+w_1t+w_0=0$ where the w_i are constants whose values are obtained by plugging in the values of P_1, G, c_1, h, C, and D into simple (but tedious to write out) formulas obtained by straightforward application of calculus and algebra (see Figure 8).

BPSolver makes use of the local minimum found by BPCompute to obtain the globally minimal solution by iterative coordinate descent. We prime our procedure with the previously obtained solution for k-1 points. We introduce the k-th point. The initial value for this point is determined by a call to BPCompute. After each call to BPCompute, the points on the prior and next boundaries (not including the start and goal points which are fixed) are added to a ChangeSet for re-evaluation in the next iteration. Points in the ChangeSet are revised by calls to BPCompute. The iterations continue until the improvement of overall cost function is less than some given threshold (or a maximum number of iterations are reached or the ChangeSet is empty). Mitchell and Papadimitriou [MP91] have noted that due to the convexity of the cost function, iterative coordinate descent procedures such as this will converge to the global optimum, but the bound on the number of iterations required to achieve a solution within epsilon of optimal are not known. They cite some empirical results that suggest that convergence will typically be fast in practice.

5. Using BP Solver in A*

Armed with BPSolver, our next task is to integrate it into the A* search algorithm. Recall the boundary points as nodes search graph representation described at the start of Section 3. To avoid the combinatorial growth in the search tree that results from the high branching factor in this representation, we instead use boundaries as nodes in the search graph. The twist we add is that boundary nodes keep track of the optimum boundary point so far. Of course, this point will change as the A* search progresses and the cost of subsequent regions replaces the heuristic estimate in the cost function.

The A* search begins in the usual way by placing the start node on the OPEN list. We treat the start point as a point on a boundary of length 0. Boundaries are treated as two sided and directed (connecting a from- and a to-region). The start node is dynamically inserted into the search graph with its to-region being the polygon that contains the start point. The goal node is treated similarly. The cost function for the start node is computed (simply 0+h|G-S|). When a node is selected from the OPEN list for expansion, its successors are boundaries of the node's to-region. A node is created for each such boundary. At this point, BPSolver is called to determine the optimal point on the node's boundary. As a consequence, we have also determined the node's f-score. These newly evaluated nodes are inserted into the OPEN list. From here, the A* search proceeds in the usual fashion.

There is a complication that must be handled. In typical implementations of A*, the f-score for a child is determined by summing the g-score of the parent and the h-score of the child. This will not work here because the g-score of the parent may be unrelated to the path determined by BPSolver for the child. This is because the boundary points used in calculating the g-score of the parent may be revised by BPSolver in the child when taking into account the actual cost of the next region that replaces the previous heuristic estimate. In fact, the parent of a node is in the search tree is not really the node's parent in the usual sense.

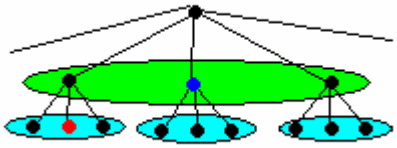

Figure 9: Boundary points as nodes

To see this, compare the boundary points as nodes representation (BPrep) in Figure 9 versus boundaries as nodes representation (Brep) in Figure 10.

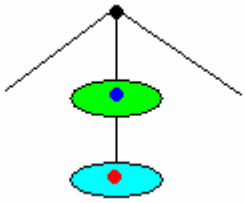

Figure 10: Boundaries as nodes.

The figures depict part of the search tree. The green ellipse encloses nodes corresponding to a boundary. The cyan ellipse(s) enclose(s) nodes corresponding to a neighbouring boundary. In Figure 9, because points are nodes, each boundary has multiple nodes.

But in Figure 10, boundaries are nodes so there is just one node per ellipse. Now consider the points in Figure 9 indicated in blue and red. At some stage of the search, the blue node in the green boundary may appear most promising. At a later stage in the search we may arrive at the red node. The parent of the red node is not the blue node (but a node corresponding to a different point on the same boundary). In Figure 10, since the green boundary is the parent of the cyan boundary, at one point of the search, the node corresponding to the green boundary may compute its g-score using the blue point. At a later stage, when we arrive at the cyan boundary, the red point is determined to be optimal using a point on the green boundary that may be different from the blue one. But because the parent of the cyan boundary may be shared with other boundaries, we cannot simply revise the point and g-score of the parent node. Instead, once we arrive at the cyan boundary, the 'parent' node is irrelevant. We must maintain the relevant information in the child node. This means that nodes in Brep will consume more space than nodes in the BPrep (but this will be offset by the fact that there will be many fewer Brep nodes). This also affects how we are to deal with cycle checks---some additional bookkeeping is required over the usual checks against the OPEN and CLOSED lists. Apart from these implementation details, the rest remains as in typical A* implementations.

6. Empirical Results

To evaluate the performance of our proposed approach, we have implemented A* using Brep and the BPSolver. For comparison, we have implemented BPrep as well. For test problems, we used a Voronoi Diagram generator [Kov99] to produce maps covered by convex polygons. This is not ideal as the convex polygon covers generated in this way are a special case of general convex polygon covers. However, this serves as a preliminary proof of concept test.

There are a number of parameters that affect performance of the two representations and these must be varied in turn to understand their effect. The first parameter is the map size. Our eventual goal is to use maps in the Torque Game Engine [Fin05]. In this engine, maps are composed of infinitely repeating terrain blocks where the size of a block is 65536 world units (about 1664.6 meters). If we scale this to 600 by 600 to fit nicely on the screen, this means every pixel is about 2.8 meters wide. This is about the resolution level we would ideally like for our paths.

For this desired resolution, BPrep would need a boundary point at every pixel along each boundary. In our experiments, we vary the boundary point spacing from 1 pixel (about right) to 32 pixels (too much). The next parameter is the number of regions the map is divided into. We experimented with 8, 16 and 24. The next parameter is the epsilon cutoff used BPSolver to terminate the descent. At present, we do not have a clear idea for choosing this parameter. In the experiments so far, we have tried, 1, 0.1, 0.01 and 0. BPSolver is also designed to terminate if it reaches a set maximum number of iterations. We tried maximum iteration set at 2, 4, 8, 16, 32, 64 and 128. We measured the path cost, time, nodes expanded and search steps used by each method.

Our preliminary findings suggest the following. The BPrep approach has difficulty at low boundary point spacing. Not unexpectedly, its speed increases and path quality decreases as the boundary point spread is increased. The Brep approach outperforms BPrep on maps with fewer regions on all settings. As the number of regions increases, Brep begins to lose on time at wide boundary point spread settings. But BPrep loses on path quality. More extensive experimentation and analysis is required to be able to make stronger claims. In this paper, we provide only some examples to indicate the potential of the approach and suggest further study is warranted.

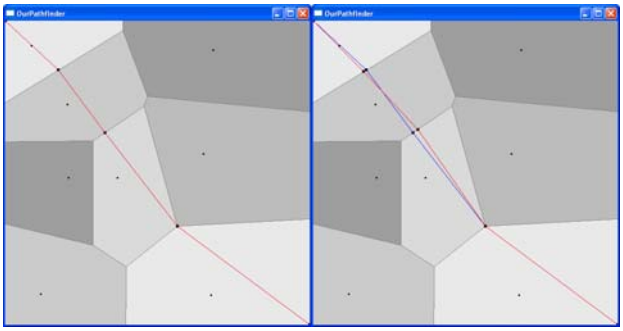

Figure 11: 8 Regions, BP spread 1 vs. 32

Figures 11 and 12 are screenshots from experiments that compare BPrep (red) and Brep (blue) on a map with 8 regions (Figure 11) and a map with 24 regions (Figure 12). The regions are shaded according to their costs with darker shading indicating higher cost. The costs were randomly generated values between 1 and 8. In all these experiments, the number of iterations for Brep was limited to 32 and a cut-off epsilon of 0 was used. The left screen shots

correspond to a boundary point spread of 1 for BPrep. The right screenshots are for a boundary point spread of 32.

In Figure 11 (left), the paths found by the two methods are identical and have a path cost of 1302.4. However, Brep required only 0.02 seconds to find this path, whereas BPrep ran for 795.66 seconds. Brep expanded 43 nodes while BPrep expanded 2,287 nodes. In Figure 11 (right), since the parameters used for Brep were unchanged, its results are the same. The path found by BPrep using the wider boundary point spread was slightly longer, costing 1306.12. The time used was fairly close (0.06 seconds). The nodes expanded were 82.

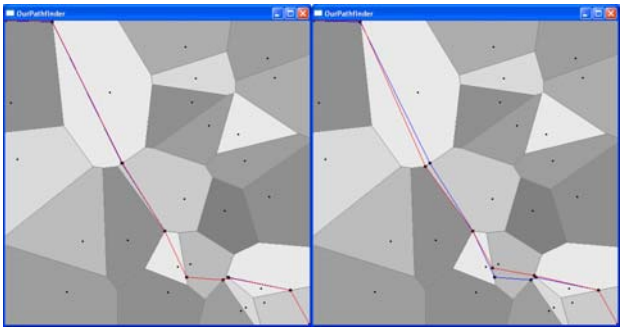

Figure 12: 24 Regions, BP spread 1 vs. 32

In Figure 12 (left), both methods find essentially the same path with a path cost of 2057.06 for Brep and 2057.08 for BPrep. Brep took only 0.41 seconds to find this, but BPrep took 4,579.8 seconds. Brep expanded 363 nodes whereas BPrep expanded 8,338 nodes. In Figure 12 (right), BPrep found a path costing 2065.72 (slightly worse than Brep) in 0.25 seconds (slightly faster than Brep) while expanding 309 nodes. While the performance of BPrep at a boundary point spread of 32 is reasonable, recall that for our goal of using Torque maps, 32 scaled by 600 means path points can be off by close to 500 meters in BPrep. Whether this is acceptable will depend on the game.

These experiments are quite preliminary but are sufficient to indicate the potential of the approach developed here.

7. Future work

Regardless of the approach taken, eventually it will bog down as the map size or number of regions is increased. To combat this, a number of hierarchical approaches have been developed [BMS04, SB06]. Hierarchical abstraction, in search, problem solving, and planning, works by replacing one state space by another (the 'abstract' space) that is easier to search. The results of the search in the abstract space are used to guide search in the original space [HZD96].

It would be interesting to integrate our method in a hierarchical approach. In fact, to a degree it will be absolutely necessary. This is because our approach relies on uniform cost regions. In the typical maps, even open areas are not completely uniform. A field that contains some isolated rocks and trees can be treated as a uniform cost region at a higher level of abstraction and paths around the trees and rocks can be resolved at a lower level of the hierarchy.

Expanding our search mechanism to allow for co-operative pathfinding in multi-agent based scenarios would also be an interesting problem. Some approaches have been identified to solve the problem of co-operative pathfinding in grid based maps [Sil06]. We believe that further optimizations are possible using our approach.

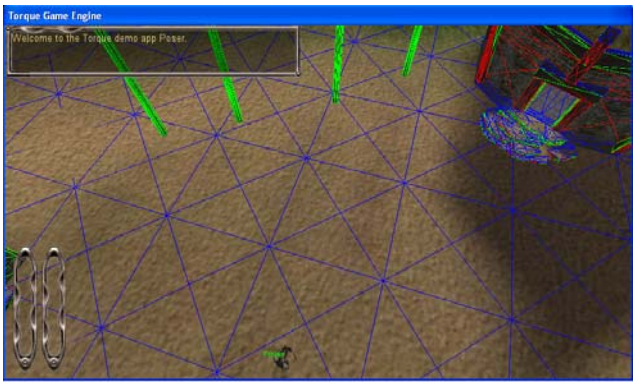

Figure 13: Torque Game Engine

We also intend to integrate the approach in the Torque Game Engine. We are working on generating a NavMesh [Toz02] representation from the world geometry. Figure 13 shows the NavMesh for a map prior removing obstacles from the mesh. This would form the lowest level of an abstraction hierarchy. Next we need to automatically generate uniform cost regions from the NavMesh. Using navigation meshes to model terrain would also allow us to work with 3D environments which we believe could be solved using a similar framework.

8. Conclusion

Our experiments with this approach to solve the problem of pathfinding in open terrain having variable cost factors and its integration with a search

mechanism based on the A* algorithm have led us to a working algorithm that efficiently produces near optimal paths. We believe that it is a viable method to determine highly precise paths that are unconstrained by the resolution issues surrounding terrain representations that use small regions as graph nodes. The trade-off of runtime speed and optimality of the path may also be controlled by modifying the resolution constant which controls the number of repair iterations run. Also worth mention is the fact that the terrain representation and node resolution used greatly affects the efficiency of traditional graph based search methods while our approach is independent of these. We believe that approaches like ours, that are alternatives to traditional rigid node based graph theoretic search techniques, offer viable solutions to the pathfinding problem and demand greater attention and research.

9. Acknowledgements

This work was funded by the first author's Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant (122131). The third author is also supported by a NSERC PGSD. The authors acknowledge valuable discussions and feedback from the members of the Centre for Interactive Digital Entertainment Research (CIDER). We also thank Dr. Myron Hlynka for providing a mathematician's perspective.

References

[BMS04] A. Botea, M. Müller and J. Schaeffer. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*. 1(1):7-28.

[CSU97] D.Z. Chen, R.J. Szczerba and J.J. Uhran, (1997). A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment. *IEEE Transactions on Robotics and Automation*. 13(5): 668-681.

[DP85] R. Dechter and J. Pearl. (1985). Generalized best-first search strategies and the optimality of A*. *Journal of the Association for Computing Machinery*, 32(3):505-536.

[Ebe05] D. Eberly. (2005). *3D Game Engine Architecture*. Morgan Kaufmann Publishers.

[Fin05] K.C. Finney. (2005). *Advanced 3D Game Programming All in One*. Thomson Course Technology.

[HNF68] P.E. Hart, N.J. Nilsson and B. Raphael. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100-107.

[HZD96] R.C. Holte, M.B. Perez, R.M. Zimmer and A.J. Mac-Donald. (1996). Hierarchical A*: Searching abstraction hierarchies efficiently, *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 530–535.

[Kov99] V. Kovalcik. (1999). *Voronoi Diagram Demo*. http://www.fi.muni.cz/~xkovalc/zip/voronoi.zip

[MP91] J.S.B. Mitchell and C. H. Papadimitiou. (1991). The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the Association for Computing Machinery*, 38(1):18-73.

[NAH04] R. Niewiadomski, J.N. Amaral, and R.C. Holte, (2004). A performance study of data layout techniques for improving data locality in refinement-based pathfinding, *The ACM Journal of Experimental Algorithmics* 9(1.2):1-28.

[Rab00] S. Rabin. (2000). A* speed optimizations. *Game Programming Gems*. (M. DeLoura, ed.). Charles River Media. 272-287.

[RN03] S. Russell and P. Norvig. (2003). *Artificial Intelligence: A Modern Approach.* 2nd Ed. Prentice Hall.

[SB06] N. Sturtevant and M. Buro. (2006). Improving Collaborative Pathfinding Using Map Abstraction. Artificial Intelligence and Interactive Digital Entertainment. 80-85.

[Sno00] G. Snook. (2000). Simplified 3D movement and pathfinding using navigation meshes. *Game Programming Gems*. (M. DeLoura, ed.). Charles River Media. 288-304.

[Sil06] D. Silver, Cooperative Pathfinding. (2006). *AI Game Programming Wisdom 3*. (S. Rabin, ed.). Charles River Media. 99-111.

[Sto00] The Basics of A* for Path Planning. (2000). *Game Programming Gems*. (M. DeLoura, ed.). Charles River Media. 254-263.

[Toz02] P. Tozour. (2002). Building a near-optimal navigation mesh. *AI Game Programming Wisdom*. (S. Rabin, ed.). Charles River Media. 171-185.

[Toz04] P. Tozour. (2004). Search space representations. *AI Game Programming Wisdom 2*. (S. Rabin, ed.). Charles River Media. 85-113.

[Toz05] P. Tozour. (2005). *Search algorithms and search space demo 1.0*. http://www.ai-blog.net.

[Yap02] P. Yap. (2002). Grid-based pathfinding. *Advances in Artificial Intelligence* (proceedings of 15th Conference of the Canadian Society for Computational Studies of Intelligence). 44-55.

[YSSB98] A. Yahja, A. Stentz, S. Singh and B. Brumitt. (1998). Framed-quadtree path planning for mobile robots operating in sparse environments. In Proceedings, *IEEE Conference on Robotics and Automation (ICRA)*, 650-655.