

Dynamic Path Planning and Terrain Analysis for Games

Paul Brobst, Ramesh Saran, Michael van Lent,

University of Southern California
Institute for Creative Technologies
13274 Fiji Way
Marina del Rey, CA 90292
[brobst, saran, vanlent]@ict.usc.edu

Abstract

Most commercial games on the market today embed into their game maps a great deal of information. Almost all of this information is precomputed and/or manual entered by the game developers. These static map annotations don't and can't take into account the many dynamic factors that can influence path planning and terrain analysis. In some cases game designers work around these issues by, for example, limiting the game to a very few deformable terrain features. In other cases the artificial intelligence systems of the game simply have to make due with approximated static terrain annotations. This paper describes two of the primary challenges in dynamic terrain, path planning and terrain analysis, and describes some early ideas for how to address these challenges.

Introduction

Most games involve some type of path planning and some type of terrain representation. However, in most cases much of this is pre-computed and encoded into the AI system's behaviors. For example, many games embed tags into the terrain representation providing the AI with hints that assist with planning paths, locating cover positions and identifying choke points. Similarly, many games pre-compute certain paths that will be frequently encountered. However, since this terrain reasoning and pathing is pre-computed it is necessarily static and can't take into account dynamic considerations such as deformable terrain features and the current position of enemy and friendly forces. Dynamic terrain reasoning takes all of these rapidly changing aspects into account and reasons about pathing, spatial awareness, and terrain features at runtime.

The main advantage to the static, pre-computed approach is the savings in runtime processing which is traded for higher memory requirements. The two approaches described here, dynamic caching for path planning and dynamic terrain analysis, seek to add a dynamic element to terrain reasoning while keeping the runtime processing and memory requirements reasonable.

Dynamic Caching for Path Planning

The industry standard for low level path planning is to use A* or Iterative Deepening IDA* search (Stout 1996) over embedded map information, such as a node network or navigation mesh (O'Neill 2004). Heavy precomputation is often done, storing connection lists in large databases in order to make queries as fast as possible at runtime. There are two distinct disadvantages to this approach. First, it uses large amounts of memory storing entries that might not ever be used. Second, it precludes the ability to have a dynamic environment; an environment that changes during the course of the game. Dynamic map information is one of the major challenges in implementing deformable terrain features in games today. If the environment changes the entire precomputed cache must be recomputed. This performance hit is unacceptable in most cases and the end result is that dynamic environments are avoided altogether. This presents a limitation to game and level design.

Dynamic caching, such as is used in processors and web caching systems, would solve these problems. Simply enough, the cache would start out empty and as valid paths are found they would be entered into the cache. The cache size could be limited and seldom used entries thrown out if it grew too big. A search algorithm would check the cache each iteration looking for a hit. If a hit is found the cache will serve up the answer and search will halt. If the cache misses the search continues normally.

There are two types of redundancy that caches can take advantage of, temporal redundancies and spatial redundancies. Both can apply to search algorithms. Often agents are moving in packs and these packs are often given orders to move to the same location. This is temporal redundancy; many of the same queries are being requested in a very short amount of time. Caches satisfy this inherently, if a matching copy is found then it is served up and in most cases only the first searcher must incur the cost.

Spatial redundancy is more complicated and we will have to sacrifice some optimality to take advantage of it. First, we can check the cache for hits not only on the initial call but also within the search algorithm. For example a

cache check could be performed whenever a node was popped off of the open queue in A*. This does, however, affect the optimality of the algorithm. A second way to take advantage of spatial redundancy would be to store not only the beginning and end of a valid search into a cache entry but also a number of nearby nodes linked off of them. This would allow a search that encountered any one of these intermediate nodes to finish at that point and use the cached solution. Depending on when in the A* algorithm the cache was checked this approach may result in a non-optimal path. There are interesting research challenges involved in exploring the trade off between speed and memory involved in different cache sizes and caching different numbers of intermediate points. There are also interesting challenges in determining when optimality should be traded for efficiency.

How does dynamic caching solve the problem of dynamically changing environments? The simple and most straightforward solution would be to invalidate the entire cache whenever a node or link is added or removed. Also, due to a dynamic cache's smaller size an algorithm to check through the cache and detect entries that would be affected would be more plausible than it would be for precomputed caches.

Dynamic caching presents benefits as well as sacrifices over fully precomputed solutions. As discussed, it provides the ability to handle dynamic networks while precomputing fails to do so. It also can provide speed savings in comparison to systems that are doing no caching at all. However it will not be able to match the runtime speed of precomputation. Dynamic caching, with the right parameters, looks to be a good middle ground for performance vs. flexibility in path planning.

Dynamic Terrain Analysis

As AI characters lack vision capabilities in a game environment, there is a need to provide spatial awareness to them by using terrain analysis.

The most common approach to terrain analysis in games is to manually annotate the terrain with hints to the AI system about the tactical suitability of the various locations for both attacks and defenses. Some of the location hints provided to the AI system include cover locations arrayed along walls and other barriers, exit references for each door into and out of a room, scramble positions that indicate where to run to when fired upon, and fixed networks for obstacle avoidance. These hints are manually placed into the game map often using 3D editors such as Maya or 3ds max. This approach is an extremely tedious process requiring the game level designer to manually identify location hints and individually placing them on the game's map structure. While this works for small maps, this quickly becomes tedious for larger terrains.

While AI annotations on the terrain provide the advantage of minimizing runtime computations, they remain valid only if the NPC movement and the terrain changes are constrained. It fails to consider the dynamic aspects of the terrain like vehicles, obstacles like barrels,

bridges which can be blown apart, minefields which can go off during the game execution. It also fails to consider the locations of enemy forces, their line-of-sight visibility and lines of fire. Force-based terrain analysis is a very challenging problem because of the mobility of the forces, which could often invalidate the pre-computed analysis results.

Influence mapping techniques remain another popular way to provide dynamic terrain information to game AI. But, they provide only a crude way to assess dynamic terrain and have scalability limitations caused by heuristics overload when too many influences affect a map.

Van der Sterren (van der Sterren 2001) has developed some ways to translate tactical terrain information into waypoint related annotation data. While these simple methods help in attaining a certain level of automation, they do not address the need to identify higher level terrain features automatically. Pottinger (Pottinger 2000) explains some of the techniques used in the Age of Empires game for zone decomposition using convex hulls.

One promising approach for automating much of the terrain analysis process is to use a geo-spatial information system (GIS), like the ArcGIS suite of products (ESRI 2004), to do the analysis and feature extraction from raw terrain data. The terrain could be divided into zones of similar terrain characteristics. The impassable terrain could then be separated from passable areas. The mobility corridors could then be identified, which leads to identification of various avenues of approach. The avenues of approach would give hints to the AI system to choose suitable directions to attack or defend depending on mission goals. A secondary benefit of dividing the terrain into zones of passability is that it aids in constraining the search space for path finding subsystems, resulting in significant speedups in path calculations.

Some common features in geo-spatial information systems are converting features to grids; generating density maps from point features; creating continuous surfaces from sample data measurements; creating contour, slopes, aspect maps and hill shades of the surfaces; performing neighborhood and zone analysis. These features would directly help in performing development-time static terrain analysis for games.

While GIS tools could help in automating static terrain analysis, none of them in their current form address the needs of dynamic terrain analysis. Unlike static terrain analysis, where most of the analysis could be done during game development and the pre-computed results being used by the game AI, dynamic terrain which must be computed at runtime poses significant challenges for interactive virtual environments like games.

One desirable characteristic of a dynamic terrain analysis solution would be to use the static analysis results as much as possible and minimally modify them to accommodate the game dynamics. One way to achieve this ideal would be to develop representations and access methods for the terrain which would accommodate partial patching of analysis results. We would also want to filter the changes in

the game world to the suitable granularity corresponding to our game AI's reaction times.

While GIS tools would definitely be useful in automating the generation of annotations, other challenges that game developers face are related to indexing, querying and updating them. Multidimensional spatial data indexing and access methods have been studied for more than two decades and could provide promising approaches towards a dynamic terrain representation with minimal effort involved in patching static analysis results. Bounding box methods like R+ trees and subdivision schemes like quadtrees and their higher dimensional versions are two popular approaches in building spatial database systems (Gaede and Gunther 1998). Unlike relational database systems, where the operations and calculi are standardized and the operators have closure property, spatial database systems are still built on application dependent needs and do not yet have standard operations, calculi and query structures. Using the spatial database system as an embedded application in games would lead to new possibilities for obtaining accurate real-time information about the terrain.

Future Work

Both dynamic caching for A* search and dynamic terrain analysis seek to move game AI systems away from static, precomputed terrain reasoning towards more intelligent, dynamic reasoning that takes many more features into account. In addition to the significant computational challenges there are a host of interesting research challenges.

Dynamic caching for A* search needs to be implemented and tested before its value can be truly gauged. Given an environment what is the optimal cache size? How long should entries live in the cache before dying out? How many intermediate links between the beginning and end points should be stored in an entry? Is it beneficial to include estimated remaining search time in a heuristic, with cache hits being a time of zero? All of these questions need to be answered before the value dynamic caching brings can be known. Fully preprocessed caches will always be faster than dynamic caching but they lack flexibility. As we move forward demand for better AI will continue to increase. With that in mind, flexibility, as well as speed, will be needed.

For terrain analysis the current tool infrastructure built for GIS products needs to be evaluated to identify their suitability in performing static terrain analysis to obtain information like mobility corridors, avenues of approach and identify directional properties like visibility, cover, camouflage and concealment offered by the terrain. Could we justify the efforts involved in automating terrain analysis in terms of the time saved by the level designer, which could be spent in more productive pursuits? This is a vital question that we intend to address by experimental evaluation.

Most of the current literature on spatial database building methods address the popular application needs like GIS and hence needs to be reexamined to identify relevant heuristics for dynamic terrain analysis in games. An in-game spatial database system to keep track of dynamics of terrain caused by movable obstacles and active entities like threat forces and suitable representations for accommodating dynamic terrain changes with minimally modifying results of static terrain analysis need to be implemented to evaluate the effectiveness of dynamic terrain analysis.

References

- van der Sterren, W. 2001. "Terrain Reasoning for 3D Action Games", *Game Developers Conference*, March 2001.
- Pottinger, D. C. 2000. "Terrain Analysis for Real-time Strategy games", *Game Developers Conference*, 2000.
- Gaede, V. and Gunther, O. 1998. "Multidimensional Access Methods", *ACM Computing Surveys*, Volume 30, Issue 2, 1998.
- O'Neill, J. 2004. "Efficient Navigation Mesh Implementation", *Journal of Game Development*, 2004, 1(1).
- Stout, B. 1996. "Smart Moves: Intelligent Path-Finding", *Game Developer's Magazine*, October 1996.
- ESRI 2004. ArcGIS Product Webpage. <http://www.esri.com/software/arcgis/> February 19, 2004.