8.5

Level-Of-Detail AI for a Large Role-Playing Game

Mark Brockington—BioWare Corp.

markb@bioware.com

The initial design document for BioWare's multiplayer title, *Neverwinter Nights* (*NWN*), called for adventures that spanned a series of *areas* similar in size to those in our previous single-player role-playing games (the *Baldur's Gate* series). Each area within an adventure is a contiguous region of space that is linked to other areas via area transitions. The interior of a house, a patch of forest, or a level of a dungeon would each be represented as a single area.

We found that we had to limit the number of resources taken up by the game and AI by limiting the engine to one *master area group* in *Baldur's Gate*. A master area group was a subset of all of the areas within the adventure. A master area group usually consisted of an exterior area and all of the interiors of the buildings within that exterior area. When a player character (*PC*) reached an area transition that would transfer it to a different master area group, all users heard "You must gather your party before venturing forth." This was a major headache during a multi-player game unless you were operating as a well-behaved party that traveled as a group. It regularly appears on our message boards as one of our most hated features.

A major design goal for *NWN* was that it must be a fun multi-player role-playing game. Thus, it was evident that restricting players' movement by the methods we used in *Baldur's Gate* would be impossible. After a brief discussion or two, we decided that we would have all of the areas within a given adventure active and in memory at all times, and reduce the size of our adventures so that they were a bit smaller in scope.

However, this posed a problem for the AI programmer. Depending on the number of areas, and the number of autonomous agents (or *creatures*) within each area, we would have thousands of agents operating within the game world. Each of these creatures could make high CPU demands on the AI engine (such as movement or combat) at the same time. Everything in the world could come to a grinding halt as the AI worked its way through thousands of requests.

Graphics engines face a similar problem when they must render a large number of objects on the screen. Since every graphics chip has a limit to the number of triangles that it can render each frame, one solution is to draw the same number of objects, but with some of them containing fewer triangles. Depending on the scene complexity,

each object can be rendered with an appropriate level-of-detail. Thus, this level-of-detail algorithm can control and adjust the number of triangles drawn each frame while still managing to draw every object.

If it works in graphics, why not for our artificial intelligence needs? We are going to illustrate the level-of-detail AI system that we implemented in *Neverwinter Nights*. We start by translating the level-of-detail concept to AI. Then, we describe how one can classify agents by the level-of-detail they require. Finally, we show how actions with high resource usage can be limited.

Level-Of-Detail from the AI Perspective

The advantages of storing 3D objects at different resolutions or levels-of-detail was first discussed 25 years ago [Clark76]. An object that is onscreen and close to the camera might require hundreds or thousands of polygons to be rendered at sufficient quality. However, if the object is distant from the camera, you might only need a simplified version of the model with only a handful of polygons to give an approximate view of what the object looks like. Figure 8.5.1 shows an example [Luebke98] of how 3D graphics might simplify a 10,000-polygon lamp into something barely resembling a lamp with 48 polygons. In Figure 8.5.2, we see an example of how to use these lower-polygon models to render four lamps, using fewer polygons than if we only had the single high-polygon lamp.

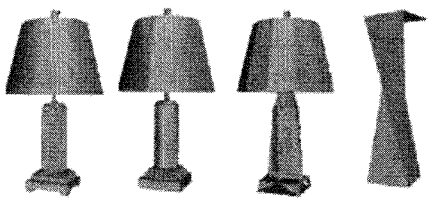

FIGURE 8.5.1 *Lamps of varying levels-of-detail, viewed from the same distance. (© 2002, David Luebke. Reprinted with permission.)*

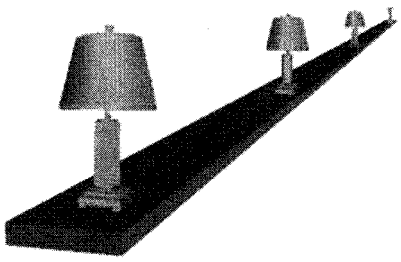

FIGURE 8.5.2 *Lamps of varying levels-of-detail, viewed at different distances. (© 2002, David Luebke. Reprinted with permission.)*

How does this translate to artificial intelligence? If we view artificial intelligence as creating the illusion of intelligence, the goal of the game AI would be to make the objects that the player (or players) can see exhibit smart behaviors. We want to perform more precise AI algorithms and techniques on them to enhance the illusion of intelligence.

The converse is also true. If an object is off screen, or is so distant that it is difficult to analyze its behavior, does it really matter whether the object is doing some clever AI? We could save CPU time by using approximations of smart behaviors on objects that no player can see.

Level-Of-Detail Classifications

What can each player see? In *NWN*, the *player character* (*PC*) is centered in the middle of the screen. The camera can be freely rotated around the PC. To control the number of objects on screen, we limited the pitch of the camera so that one could see at most 50 meters around the character at any one time.

There are secondary factors to consider when classifying objects. Your best algorithms should go into controlling player characters, since they are always on screen. If players do notice intelligent behavior on other creatures, the players will be looking for it more closely on creatures that are interacting with their PC (either in combat or in conversation). Finally, if a creature is in an area in which there is no player to see it, a further relaxation of the AI is possible.

In *NWN*, there are five different classifications for an object's level-of-detail, as shown in Table 8.5.1 going from highest to lowest priority.

Table 8.5.1 LOD Levels in *Neverwinter Nights*

LOD	Classification
1	Player Characters (PCs) (your party)
2	Creatures fighting or interacting with a PC
3	Creatures within 50 meters of a PC
4	Creatures in the same large-scale area of a PC
5	Creatures in areas without a PC.

In Figure 8.5.3, we see a number of objets within two separate areas. Each large square in Figure 8.5.3 represents a different area. There are three players in the game, each controlling a single PC. PCs are represented by a small circle, with a large circle indicating the area that can be seen by the player. Creatures are represented with a small square. Both PCs and creature symbols contain their current LOD.

PCs are always LOD 1, so each small circle contains a 1. Creatures fighting or interacting with PCs contain the number 2. Creatures in the proximity of PCs contain the number 3. Creatures that are not within a 50-meter radius of any player

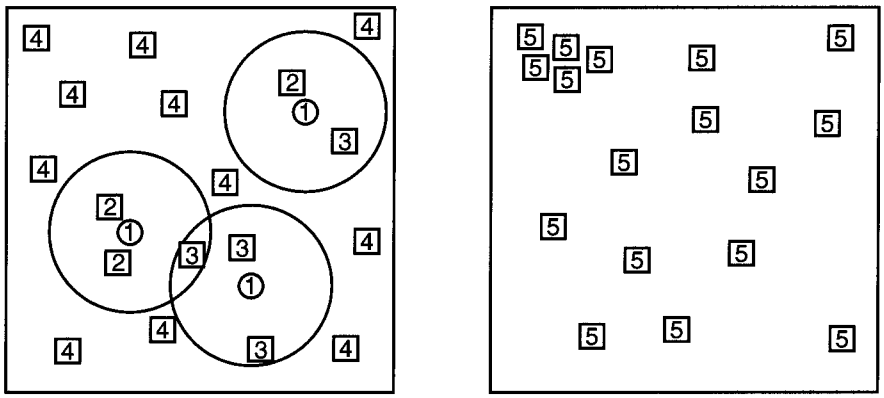

FIGURE 8.5.3 *The five classifications of level-of-detail in* NWN.

contain the number 4. Meanwhile, all of the creatures in the second area contain the number 5, to represent that they are at the lowest level-of-detail classification.

Each of these levels is easy to detect via mechanics that are already set up within the game engine. For example, each creature (not just the PCs) attempts to do a visibility check to all creatures within 50 meters of itself, and it is straightforward to determine if any of those nearby creatures are PCs. Each area has a scripting event that runs when a PC moves out of an area and/or into another area (whether it is by connecting to the game, performing an area transition, or disconnecting from the game). Thus, determining and maintaining which areas have PCs in them is easy. All attack actions go through a single function. This is beneficial, as it means there is only one spot to check if the creature is attacking a PC and, hence, promoting the attacking creature to the second highest level of priority.

Using the Level-Of-Detail Classification

There are five level-of-detail (LOD) classifications that determine update frequency of a creature.

Which Creature Gets the Next Time Slice?

The perceived intelligence of a creature is strongly correlated to its reaction time in specific situations. For example, a creature that takes too long to chase after a fleeing PC does not seem as intelligent as a creature that immediately follows the PC as it runs away. Thus, the five LOD classifications should determine the update frequency in order to maintain the required level of interactivity or intelligence.

In *NWN*, each of the creatures is placed on one of the five lists. After we have categorized all AI objects on to one of the five lists, a function then determines who gets the next time slice. In *NWN*, we found that the percentages in Table 8.5.2 work well.

Table 8.5.2 Percentage of CPU Time for Each Level-of-Detail

CPU Time	Level-of-Detail
60%	LOD 1: Player Characters
24%	LOD 2: Creatures interacting with PCs
10%	LOD 3: Creatures in proximity of PCs
4%	LOD 4: Creature in an area with a PC
2%	LOD 5: Creatures not in an area with a PC

When an LOD list is allowed to execute, the first creature waiting on that list's AI queue is processed. Each creature from that list is processed one at a time until the list's time slice runs out. Each list is not allowed to steal time from the processing of the lower-priority lists. This guarantees that at least one creature from each list is allowed to run an AI update, and prevents starvation of the lowest-priority list.

How Pathfinding Is Affected by Level-Of-Detail

Many AI actions can use up CPU cycles. One of the most time-consuming actions that one can perform is pathfinding over a complicated terrain. Thus, a pathfinding algorithm is a logical place to start implementing a level-of-detail algorithm.

Based on the structure of the AI, we decided not to store the pathfinding progress in the game world data structure. Instead, each creature keeps its own instance of how the pathfinding algorithm is progressing. However, using an A* approach in this manner would be extremely expensive; we could have dozens of agents attempting to perform pathfinding on the same area at the same time, with each pathfinding search requiring a large amount of RAM. In *NWN*, we actually used IDA* and other computer chess techniques to save on the resources used [Brockington00].

The terrain is composed of 10-by-10 meter tiles, each of which has information indicating how it interconnects between other tiles. With this coarse-grained system (*inter-tile pathfinding*), one can easily generate a series of tiles that one must travel through to reach the final destination. We generate our full path by using *intra-tile pathfinding* over the 3D geometry on each tile specified by our inter-tile path.

How can one take advantage of this structure with the level-of-detail algorithm? Creatures that are on screen are always required to do both an inter-tile path and an intra-tile path to make paths look smooth and natural. Any creature at LOD 1 through 3 in our hierarchy implements the full pathfinding algorithm. However, it is important to note that the difference in CPU time allocated to creatures at LOD 1 versus creatures at LOD 3 is significant. For example, in our urban areas, each PC can usually see eight other non-player characters standing on guard, selling goods, or wandering the streets. Based on the time percentages given earlier, a PC could compute a complex path 48 times faster than a creature near a PC.

What about LOD 4 creatures? We only bother to compute their paths via the inter-tile method, and then spend the time jumping each creature from tile to tile at

the speed that they would have walked between each tile. The intra-tile pathfinding is where we spend over 90 percent of our time in the pathfinding algorithm, so avoiding this step is a major optimization. Even if we had not used this approach, the amount of time we actually devote to each LOD 4 creature is quite small, and the creatures would seem to be "popping" over large steps anyway!

For LOD 5 creatures, we do not even bother with the inter-tile path, and simply generate a delay commensurate with the length of the path (as the crow flies), and jump the creature to its final location.

What happens if a creature has its priority level increased? Well, in this case, the path must continue to be resolved at the intra-tile level (in the case of a LOD 4 creature moving to LOD 3 or above), or at the inter-tile level (in the case of a LOD 5 creature moving to LOD 4).

How Random Walks Are Affected by Level-Of-Detail

Random walking is another action that BioWare designers have used regularly on creatures in earlier titles. In short, the action takes the creature's current location, and attempts to have the creature move to a random location within a given radius of its current location.

The advantages of level-of-detail are clearer for random walks. In the case of LOD 1 or 2 creatures, we use the full pathfinding algorithm again. For LOD 3 creatures, we only walk the person in a straight line from his current location to a final location. This is accomplished by testing to see if the path is clear before handing the destination to the full pathfinding algorithm. Creatures at LOD 4 and 5 do not move based on random walk actions, because there is no point in executing the action until a PC is there to see them move.

How Combat Could Be Affected by Level-Of-Detail

In *NWN*, we use the highly complex *Advanced Dungeons and Dragons* rule set to determine the results of a combat. The base combat rules fill 10 pages of text, and there are over 50 pages of exceptions to the base rules specified in the *Player's Handbook* alone.

To work out the result of a combat in real time is a technically challenging task; one that could be optimized in our current system. At LOD 1 and 2, we have to implement the full rule system, since the results of these combats are actually shown to a PC. At LOD 3 or below, a PC is not privy to the rolls made during a combat. If there are things that are too complicated to compute in a reasonable amount of time, one does not need to compute them at this level or below. However, LOD 3 creatures must be seen to be performing what looks like the actual rules.

At LOD 4 or 5, one does not even need to use the rules; no one can see the fight, so the only thing that matters is that the end result is relatively close to what would happen in practice. Rules can be simplified by analyzing the damage capabilities of each creature, multiplying this by the number of attacks, and then randomly applying

that damage at the beginning of each round based on the likelihood of each attack succeeding. At LOD 5, one could automatically resolve combat based on a biased coin flip instead of spending the time to actually compute which character is stronger. The differences in the challenge ratings of each creature could be used to bias the coin flip.

Conclusion

In this article, we described a system for implementing AI in a role-playing game with thousands of agents. We also showed a hierarchy for classifying objects into levels-of-detail, and how to use this classification to determine which algorithm to use for some resource-intensive actions. Areas that you can exploit based on level-of-detail include:

- Processing frequency.
- Pathfinding detail, especially if you employ hierarchical searching.
- Pathfinding cheating.
- Combat rules and detail.

The actions and classification that we presented focused on discrete levels-of-detail. However, most graphics research focuses on various methods of doing continuous level-of-detail, such as progressive meshes [Hoppe96]. A couple of approaches for continuous LOD were considered for *NWN*. We experimented with varying the size and depth of the pathfinding search based on a continuous LOD measure (such as the distance to the nearest player), but our results were unsatisfactory in comparison to the discrete LOD system described in this article. We also experimented with a continuous LOD system for smoother interpolation of processing frequency, and failed to provide any additional benefit over and above what we saw with our discrete LOD system. We hope that your experiments are more successful.

References

[Brockington00] Brockington, Mark, "Pawn Captures Wyvern: How Computer Chess Can Improve Your Pathfinding," *Game Developers Conference 2000 Proceedings*, pp. 119–139, 2000.

[Clark76] Clark, James, "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, Vol. 19, No 10, pp. 547–554. 1976.

[Hoppe96] Hoppe, Hugues, "Progressive Meshes," *Computer Graphics*, Vol. 30 (SIGGRAPH 96), pp. 99–108, 1996.

[Luebke98] Luebke, David, Ph.D. Thesis, "View-Dependent Simplification of Arbitrary Polygonal Environments," UNC Department of Computer Science Technical Report #TR98-029, 1998.