# Grid Computing for Artificial Intelligence


## J.M.P. van Waveren


**May 25th 2007**



## Abstract


To show intelligent behavior in a First Person Shooter (FPS) game an Artificial Intelligence (AI) controlled character needs spatial and temporal awareness of the environment. Real-time environment sampling to acquire this awareness during game-play is particularly expensive in today's high detail games. Furthermore, on single CPU/core systems, that are still very common in the PC gaming landscape, there is only a small percentage of CPU time allocated for AI. As such, pre-calculated data structures need to be used as much as possible to allow the AI to quickly analyze the environment and make intelligent decisions. Id's new title, Enemy Territory QUAKE Wars is powered by the revolutionary *id Tech 4* game engine which has a system in place that allows the artificial players (a.k.a. bots) to quickly acquire the spatial and temporal awareness needed in order to perform well in the game. During the pre-calculation phase of this system, a boundary representation (b-rep) of configuration space (C-Space) is calculated in the form of one or more two-manifold triangle meshes. This is a complex and time consuming process where geometric operations are performed on millions of triangles. To significantly speed up this process the work is divided into many independent smaller chunks which allows maximum parallelism to be exploited through the Xoreax Grid Engine (XGE) of IncrediBuild's distributed build technology. Pre-calculation for AI in computer games is an area where the XGE is very effective in bringing down compilation times and as such in reducing the turn-around time during development.

# 1. Pre-calculation for AI

Both spatial and temporal awareness are key to performing well in an FPS game. The human brain is very good at recognition and prediction and human players take advantage of these skills to quickly become aware of the environment in an FPS game. After playing an FPS game a few times people easily recognize where they are in the environment and quickly anticipate what is going to happen, which allows them to make decisions on where to go and what they need to do. At any given time an Artificial Intelligence (AI) controlled character for an FPS game also needs to be able to quickly acquire the spatial and temporal awareness required to make an intelligent decision on where to go and what needs to be done.

Even though multi-core systems are rapidly taking over the PC gaming landscape there are still a large number of single CPU/core systems out there. On these systems the amount of CPU time allocated for AI is quite often still no more than 10 percent. At the same time the game environments are becoming much more complex and dynamic. The polygonal complexity of the environments has increased significantly over the years and in today's games there are many more dynamic objects that are moving through the environment with the use of sophisticated real-time physics simulations. In order to deal with this increased complexity while using as little CPU as possible, it is key to pre-calculate anything for the AI that can be pre-calculated. Everything that doesn't change and is static in the environment needs to be considered for pre-calculation to produce data structures that make it easier for the AI controlled characters to quickly understand their position and situation in the environment. Furthermore it is important to pre-calculate data structures that will make dealing with dynamic objects faster in real-time.

The "pre-calculated" data for the AI can be added to a game either programmatically or by hand. Certain hints for the AI are seemingly easily placed in a level by a level designer, like for instance waypoints, paths, camp spots, cover locations, etc. However, placing optimal waypoints and creating good paths such that the AI can easily reach all locations in a level is a non-trivial task. Furthermore as strategies become more complex so do the hints that need to be placed in a level. A level designer typically needs to manually place a large number of hints to cover the whole environment. Before placing hints, a level designer needs to be educated in placing these hints which takes time. A level designer can also make mistakes, especially when the hints that need to be placed are complex and/or optimal placement is not well defined. Manually placed waypoints, paths and hints require continuous testing to catch any mistakes and to generally make sure the AI is able to use the hints as intended.

An algorithmic solution to provide the AI with the information it needs may initially have problems due to bugs in the implementation. However, when a bug is fixed it usually does not

come back, whereas with a manual solution human error can be introduced at any time during development. An algorithmic solution results in scalability, repeatability and consistency. Furthermore scaling is much cheaper. Hiring, educating and employing another level designer is expensive. An algorithmic solution may initially require some programmer time but is in the end much cheaper in that more levels can be compiled in a shorter period of time by simply using more computing power.

# 2. Area Awareness System

The bots in Enemy Territory QUAKE Wars (ETQW) use an Area Awareness System (AAS) to understand, navigate and quickly become aware of the environment. This area system is automatically derived from the level geometry during an off-line compilation process. The first step of the AAS compilation process involves the construction of a boundary representation (b-rep) of configuration space (C-Space). The b-rep of C-Space consists of one or more two-manifold meshes that describe the Minkowsky sum of the world geometry and the bounding volume in which a bot (or player) resides. In the next step the AAS compiler identifies walkable surfaces on the b-rep of C-Space. The walkable surfaces are subdivided into the least number of walkable areas, such that a bot or player can move in a straight line between any two points in an area. This requires a considerable amount of filtering in order to ignore small details that should not be considered obstacles for player navigation. Next so called "reachabilities" are calculated that specify how a bot can navigate from one area to another.

The AAS implements a hierarchical routing system to find routes through the environment in real-time. This system includes a cache manager to cache routes and as such avoid frequent recalculations of routes. The hierarchical nature of the routing system makes calculating routes very fast and minimizes cache sizes. Once a route has been calculated a path optimizer is used to plot straight and curved paths through the environment along a route. The path optimizer traces along the floor to test whether or not an AI can walk to a certain point or walk in a certain direction. These floor traces are very fast because the AAS filters out all the small details that are not obstacles for player navigation but would be returned by a regular collision detection query. Once the optimized path has been calculated a system is used to calculate paths around arbitrary complex configurations of dynamic obstacles. This system modifies the optimized path if there are dynamic obstacles in the way. While calculating a new path the obstacle avoidance system also takes the static boundaries of the environment into account to keep the AI from running into static world geometry. The AAS is setup to quickly provide all local world boundaries, like walls and ledges, that need to be taken into account while avoiding obstacles.

One of the most basic and fundamental things an AI needs to be able to do is to become aware of its current location in the environment and understand the immediate surroundings. Many other systems for AI use spatio-temporal coherence where the understanding of the current location of the AI is derived from a previously known location. However, in a highly dynamic game environment events may happen that cannot always be anticipated and the AI may end up in a location far away from any previously known location. For instance the AI may have been walking close to a ledge and got pushed off, an explosion could have sent the AI flying, or the AI got dragged along by a vehicle, etc. Becoming aware of the current location in the environment (for instance by finding a nearest waypoint) through regular environment sampling without knowing a previous location is exceedingly expensive in today's high detail game environments.

The AAS allows an AI to become aware of its current location in the world at any time, with the calculation of just 10 to 20 dot products, without having to cache a previously known location. When the AI is aware of its current location the system immediately provides a lot of the information the AI needs to make an intelligent decision on where to go and what needs to be done. The system then allows the AI to quickly perform additional queries to gather any additional information it needs, like routes, paths, travel times and other properties of the environment.

# 3. Grid Computing

The construction of the b-rep of C-Space is by far the most expensive step in the AAS compilation process. For several reasons the AAS compiler is setup to divide the work involved in calculating the b-rep of C-Space into many independent smaller chunks. The math involved in geometry processing is not very complex. However, it is typically hard to anticipate all the different situations an algorithm for geometry processing needs to cover. Even when an algorithm covers all cases and works flawlessly in theory, the algorithm may not produce the desired results due to floating point rounding. Implementations of algorithms for geometry processing typically involve many lines of code to handle and deal with floating point rounding. To generally avoid a lot of floating point rounding errors and to make the handling of rounding errors easier, it is useful to process geometry in smaller chunks centered around the origin using a well defined range of floating point values. When the work is divided into smaller chunks it is also easier to analyze and debug problems during the development of the algorithm. The small chunks can be recalculated quickly and problems are localized and can be analyzed in isolation from the rest of the geometry.

Since the work is already divided into many independent smaller chunks, the construction of the b-rep of C-Space can easily be sped up by using a grid computing solution. The Xoreax Grid Engine (XGE) of IncrediBuild's distributed build technology is invaluable for grid computing solutions in that it involves minimal integration time and is very easy to use. The XGE virtualizes the file system and automatically distributes the work including the executables required to perform the work. In other words the level geometry can be modified locally and the executables can be recompiled locally without having to manually redistribute the work and executables to many computers. To get the best performance through the XGE, an executable is required that is lean, mean and loads very fast. Fortunately it is easy to split off only the absolutely necessary source code for the task at hand into a Win32 console application. There are three interfaces for queing XGE tasks, one of which is an XML interface. A simple XML file is created that lists the work and the executable to be used to process the work. The XGE xgConsole.exe command line application can then be spawned as a piped process such that the console output can be redirected to the game application or level editor.

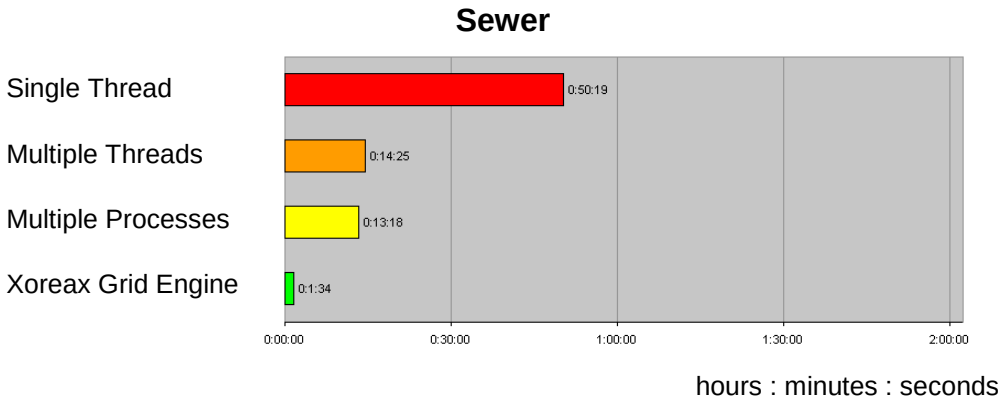
# 4. Results

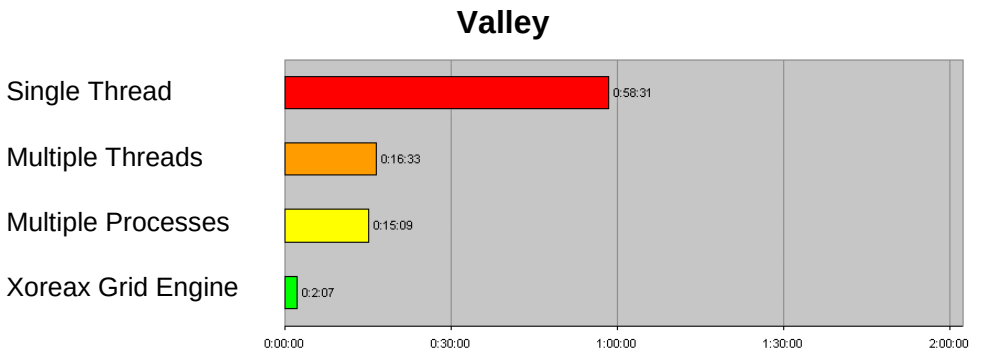
Several different approaches to exploit parallelism for the construction of the b-rep of C-Space have been implemented and tested. The algorithm is typically first implemented in a single threaded application. A natural progression is to use multiple threads. Instead of using multiple threads it is also an option to spawn multiple processes where each process does part of the work. This is very similar to what the XGE does except that only a single computer is used. The next option is obviously to use the XGE and distribute the work across many computers.

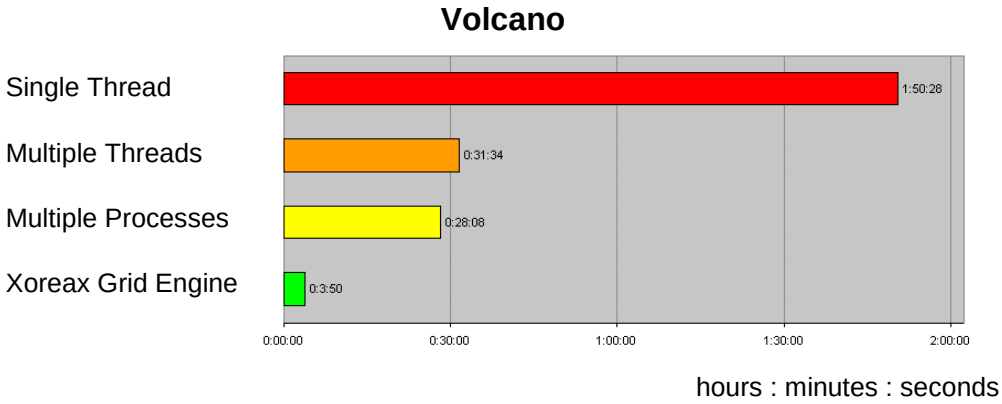
Several of the ETQW levels have been compiled using these four approaches and the compile times have been measured. Some statistics of the levels that were used are listed below. For each level the number of triangles used for player collision detection is listed. Furthermore the number of chunks with geometry is listed, where each chunk is stored in a separate file. Next the total size of all chunk files on disk is listed, and the number of triangles used to describe the b-rep of C-Space is listed as well.

| Level Statistics | | | | |
|---|---|---|---|---|
| **Level** | **# triangles** | **# chunks** | **total size** | **# C-Space triangles** |
| Sewer | 112288 | 2768 | 3.56 MB | 156328 |
| Valley | 120133 | 4241 | 4.13 MB | 167638 |
| Volcano | 217553 | 2301 | 4.91 MB | 237680 |

The charts below show the compile times for the levels using the different approaches. The Single Thread approach uses one core on a system with two Intel 2.8 GHz Dual-Core Xeon CPUs ("Paxville" 90nm NetBurst micro-architecture with HyperThreading disabled). The Multiple Threads approach uses four threads where each thread consumes one of the four cores on the same system. The Multiple Processes approach uses four processes where each process consumes one of the four cores also on the same system. The Xoreax Grid Engine approach uses a network of 14 computers for a total 78.6 GHz in 27 cores of which 9 cores are based on the Intel NetBurst micro-architecture and 18 cores are based on the Intel Core 2 micro-architecture. The computers all participate in a 1 Gb network, and as such there is minimal networking overhead because of the high speed networking infrastructure and the small file sizes.






Even though there is some overhead involved in starting a new process, using multiple processes is faster than multi-threading. The threads do not use their own separate memory pool and as such there is memory contention. This bogs down the multi-threading solution, while each process in the multiple processes solution has its own address space. The XGE solution scales pretty much linearly with the number of CPUs/cores that are available. In the above examples the XGE even scales beyond linearly with the available GHz compared to single threaded, because the cores based on the Core 2 architecture are faster than the Paxville cores.

# 5. Going Forward

As the available CPU power increases rapidly while memory and IO bandwidth do not increase at the same rate compression is becoming more important. More and more CPU power will be used to decrease storage and bandwidth requirements. From a grid computing standpoint asymmetric compression is particularly interesting, where real-time decompression during gameplay is really fast, while off-line compression may be very computationally expensive to get the best possible quality at the highest compression ratio. Grid computing lends itself very well to this kind of off-line compression. The XGE is particularly useful for sound compression and texture compression. There are typically many sounds and textures and in the case of large sounds and large textures they can be subdivided into many smaller chunks to allow massive parallelism to be exploited through the XGE.



id - defined by Freud as the primal section of the human psyche; id Software, located in Mesquite, Texas, was founded in 1991. From inception to present day, id Software has relentlessly provided technical, design and artistic leadership as an independent game developer and technology provider. Transcending the games industry, id's iconic brands such as Wolfenstein, DOOM, QUAKE and Enemy Territory have become staples of popular culture for generations of gamers. More information on id Software can be found at www.idsoftware.com.