# An Optimized Synthetic Environment Representation Developed for OneTESS Live Training

**Steven Borkman, Gregory Peele, Chuck Campbell**

**Applied Research Associates**

**Orlando, FL**

sborkman@ara.com, gpeele@ara.com, ccampbell@ara.com

## ABSTRACT

When designing a synthetic environment terrain database format, developers face a tradeoff between physical storage, runtime performance, and data accuracy. The context of the simulation and particularly its specialized requirements heavily influence how the tradeoffs are made. One of the largest historical driving factors in how this balance has been struck has been the "domain" context. The virtual and constructive training domains drove most of the modern terrain format development. However, the requirements for live training are often significantly different. For example, the OneTESS player units allow minimal storage, require a small memory footprint, and necessitate a high degree of ground truth accuracy. The requirements satisfied by existing terrain formats fail to meet these requirements.

OneTESS requires terrain resolution far beyond anything handled by previous "high end" simulations. However, OneTESS requires far fewer terrain services than traditional virtual and constructive systems. This duality makes OneTESS's extreme representation requirements attainable - the tradeoffs between time, space, and accuracy is balanced in the context of a single, high-importance function. Furthermore, OneTESS must execute on a handheld player unit possessing highly limited resources and performance capability compared to current desktop workstations.

In this paper, we discuss the OneTESS terrain requirements and the rationale for needing its own representation. We introduce a new terrain format specifically targeting the OneTESS live training and test domains. We describe its design and implementation and report the preliminary performance benchmarks of terrain services developed for this new terrain format. We conclude with ongoing efforts and future directions.

## ABOUT THE AUTHORS

**Steven Borkman** is a Senior Scientist at ARA with over 7 years of experience developing synthetic environment software for the simulation community. Mr. Borkman is currently a lead developer of the OneTESS LTF project and Principal Investigator for the OneSAF GPU Integration project. Mr. Borkman holds a Bachelor of Science degree in Computer Science from the University of Central Florida.

**Gregory Peele** is a Staff Scientist at ARA with over 2 years of experience developing synthetic environment software for the simulation community. Mr. Peele is currently a lead developer on the OneTESS LTF project. Mr. Peele holds Bachelor of Science degrees in Computer Science and Mathematics from the University of Central Florida

**Chuck Campbell** is a Principal Computer Scientist at ARA. He has over seventeen years experience developing Semi-Automated Forces (SAF) software. He has been a developer and technical lead for Synthetic Natural Environment programs including CCTT, WARSIM, OneSAF Objective System, and the Rapid Unified Generation of Urban Databases (RUGUD) database generation effort. Mr. Campbell holds a Bachelor of Science degree in Computer Science from Indiana University and a Master of Science degree in Computer Science from the University of Central Florida.

# An Optimized Synthetic Environment Representation Developed for OneTESS Live Training

**Steven Borkman, Gregory Peele, Chuck Campbell**
**Applied Research Associates**
**Orlando, FL**
sborkman@ara.com, gpeele@ara.com, ccampbell@ara.com

## BACKGROUND

Synthetic natural environment (SNE) requirements vary based on training system function. Simulation systems can run the gamut from a command staff trainer, which may require a low resolution but cover an extremely large geographic area; to an individual combatant virtual trainer, which may require a very high fidelity environment representation that correlates to the real world. Feature data such as building interiors, underground structures, and weather models may also be needed.

Systems tailor their environment requirements based on available physical resources. Limited resources lead to the traditional trade off between data storage, complexity, and accuracy. In a perfect world where time and storage space are not an issue, designers could use the most complex, highest fidelity terrain and reasoning services for their synthetic environment. Of course, such systems do not exist. This causes compromises on data complexity and model resolution in order to meet storage, accuracy, and performance requirements. The target simulation domain - live, virtual, and constructive - also greatly affects the requirements of a system.

Often space, time, and accuracy trade-offs are fairly straightforward when considered in the context of a single functional requirement. However, modern semi-automated forces (SAF) systems must support a wide range of services and functionality. And in some cases (e.g. CCTT) the same SNE services must support applications beyond SAF (manned simulators, user workstations, etc.). The need to support services as diverse as height of terrain, line of sight, and cover/concealment complicates trade offs, quite often to the extent that multiple terrain formats are created within the context of a single system. ModSAF/OTB, CCTT SAF, and OneSAF all use multiple on-disk formats to handle specialized functions (e.g. OOS stores route planning networks separate from the file containing terrain polygons).

Terrain reasoning services tend to dominate a simulation's CPU use. Consequently, SNE services and their underlying databases are often highly optimized for specialized needs. SNE developers place a high premium on performance specializations. Note that two of the major formats presently in use by Army simulations, CTDB ("Compact" Terrain Database) and MrTDB ("Model Reference" Terrain Database), are named for the primary specialization or "improvement" made for those formats. Optimization seen in applications such as CCTT and OTB SNE representations are not casual "point" improvements, but systemic and systematic specializations that permeate the code. The sheer complexity of the code and underlying formats required for these specializations illustrates how critical the optimizations were to the developers, as they sacrificed maintainability for performance and space improvements.

The synthetic environment requirements for live training systems signify a complete paradigm shift from the virtual and constructive systems. High resolution data is no longer a luxury but a necessity. Live trainers blur the line between simulation and reality in such a way that real world and virtual world correlation are paramount. To compound the problem, real time terrain reasoning functionality is essential. For the most part, live training systems have to execute on more complex, higher resolution data in a smaller performance allocation than their virtual and constructive counterparts.

Fortunately, whereas live trainers have more stringent requirements for line-of-sight (LOS) algorithms, they often have much less need for other terrain reasoning services. Since entities in the constructive domain and automated forces in virtual simulations are computer controlled, they rely on artificial intelligence to make decisions. The synthetic environment must provide the pertinent data, such as soil types and route networks, to support the decision making process. These systems must also provide advanced terrain reasoning algorithms to interpret the data. Human players in live simulations use their own senses to infer this information, allowing the environment to prioritize its effort on LOS.

In order to meet the specific needs for a simulation system, a "one size fits all" mentality generally does not work. Each simulation domain requires different data and services from a synthetic environment. Beyond that, each system architect needs to strike a balance between available resources and their system's particular requirements. A live trainer may have no need for soil types, functional regions, or route networks, but these may be essential to a constructive or virtual system. However, the live trainer may require precise correlation between the real and virtual world, which is not necessary for a constructive trainer.

The following paper addresses the SNE requirements of the OneTESS training system. It then gives an overview of common terrain representations present today in the simulation community. It addresses each representation's strength and weaknesses and their ability to meet OneTESS requirements. Finally, it presents the new LTF format and describes the design of the system.

## ONETESS TERRAIN REQUIREMENTS

Live training has historically relied on systems that simulate weapons fire and hits via active emitters and sensors. This approach has a number of weaknesses which OneTESS is addressing through their concept of geometric pairing. Geometric pairing couples a shooter with a specific target to algorithmically determine if the shooter's munition intersects its target. This is in contrast to the MILES system, which uses laser beams for pairing. In addition, geometric pairing can model Non/Beyond Line of Sight (NLOS/BLOS) whereas lasers cannot. Geometric pairing levies SNE requirements that fall far outside the design space of existing SNE representations, through a combination of requirements for extremely high-resolution terrain representation on severely limited hardware.

For OneTESS to meet its operational objectives, the terrain databases must correlate with the real world. For example, if an individual combatant seeks cover behind a ridge the simulated environment must accurately represent that ridge. The terrain traversal algorithm must precisely calculate the "electronic bullet" flight path.

This also applies to the terrain features. Some natural and man made features can provide cover. These features include trees, buildings, ditches, and foxholes. These features must also correlate to the real world to achieve an accurate simulation.

The following section highlights the major terrain requirements for OneTESS:

- To support OneTESS geometric paring functionality, the OneTESS SNE data must be on the order of 0.1 meter accuracy in x, y, and z for terrain elevation and feature data.
- In order to meet accuracy requirements, the objective resolution for the OneTESS SNE database is a 1 meter elevation grid.
- To support the small storage footprint of the OneTESS player unit, the OneTESS SNE database must be a compact format - Milestone C limitation is 5.5 gigabytes.
- To support the small memory footprint of the OneTESS player unit, the OneTESS SNE software must execute within a small memory footprint – Milestone C limitation is 55 megabytes.
- The OneTESS SNE must support real time calculations to support engagements while using an embedded CPU that has a fraction of the computing power of a desktop unit.

In order to meet its terrain requirements, OneTESS must run on a SNE capable of supporting real time terrain calculations, such as line of sight, over an extremely high resolution database. To complicate the issue, the terrain must possess a small form factor in both memory and on hard drive. The following section describes some typical terrain representations and the systems that use these representations.

## TRADITIONAL TERRAIN REPRESENTATIONS

Elevation grids and triangulated irregular networks (TIN) are the two major terrain representations found in most modern simulators. Both representations have evolved over time to support a simulator's unique requirements and feature representations.

## Gridded Terrain Representation

Early constructive databases tended to use gridded data as they come in source, a two dimensional set of elevation values, usually stored as fixed-point integers. Historically, gridded data from the National Geospatial-Intelligence Agency (NGA) was available at roughly 100m spacing. Because of the sparse terrain data, the gridded representation proved to be sufficient for some higher level command staff trainers, but not for high-fidelity lower-echelon training. Earlier SAF systems (e.g. ModSAF through the mid-90s and CCTT SAF to this day), used gridded terrain representations because they provided constant time look up, and early image generators correlated well this representation in the general case.

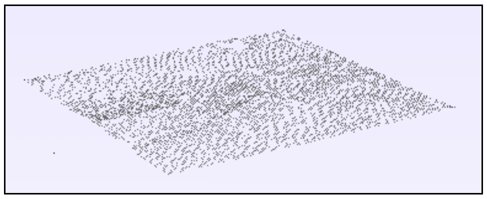

**Figure 1. Gridded terrain elevation data.**

The major flaw of the gridded terrain format is data accuracy with respect to feature placement. Microterrain was created to alleviate this issue. Microterrain is a set of smaller, irregularly shaped triangles used to characterize terrain topography that regular grids cannot represent. For example, draping a road over the terrain can lead to steep sections of road or unrealistic side slopes. Raising the road above the terrain or lowering it beneath the terrain leaves gaps between the road surface and the surrounding terrain surface. Microterrain fills the gap, creating a continuous surface over which entities can traverse. This is sometimes referred to as "cut and fill". While microterrain increases the accuracy of the database over a gridded representation, it requires greater storage due to increased triangle density and the need to explicitly store x and y location values. Microterrain also impacts runtime algorithms and performance because the areas containing irregular triangles require special handling.

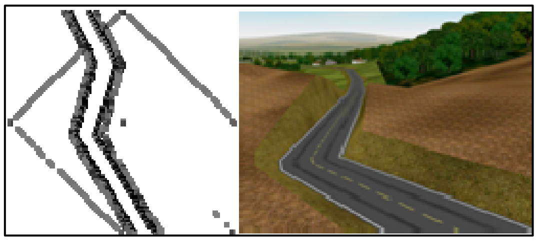

**Figure 2. An example of the use of microterrain with a regularly spaced grid.**

## Triangulated Irregular Network (TIN) Representation

Triangulated irregular terrain networks (TIN) use triangles of any size and shape to represent the terrain surface. TINs are an extension of the microterrain representation, applied to the entire database area. The theory behind TINs is to provide a best fit to the original terrain data with a minimum amount of triangles. TINs (and the ITINs described below) typically explicitly store connected topology between triangles to reduce special case code and increase algorithm performance. This approach improves correlation to the source data, but typically suffers a performance penalty because terrain services cannot leverage the regular nature of a gridded or right-triangulated representation. For high resolution data, it can also require additional memory space since the irregularity forces explicit storage of all three components (x, y, and z) of each vertex in the TIN.

## Integrated Triangulated Irregular Network (ITIN) Representation

An integrated TIN (ITIN) uses terrain feature borders as constraints to the TIN generation process. Features are typically stored separately from the terrain surface. By integrating feature boundaries into the TIN each triangle has an exact mapping to the feature covering it, whether it be a road, lake, or grassland area. ITINs typically require more triangles, due to the feature integration, but may increase performance by combining the elevation and surface material lookups into a single query. ITINs also have the secondary effect of a more complex terrain generation process.

The impact of ITINS to the terrain skin representation is an increase in the number of triangles generated. Instead of simply computing a best fit of triangles to the original source elevation data, ITINs must incorporate the vertices of integrated feature break lines during the TIN process. The resultant TIN must contain every point of each integrated feature, in addition to the triangles generated by fitting the triangles to the source elevations. An increase in triangles equates to an increase in storage requirements. This increase in triangles negatively impacts performance of terrain services that traverse the triangles.

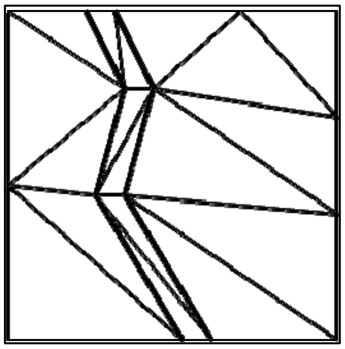

**Figure 3. Road feature integrated into the terrain skin to create an ITIN.**

## STORAGE AND PERFORMANCE FACTORS

### Round Earth Representation

Unfortunately for terrain simulations, the Earth is not flat. One driving factor for storage and performance is the trade-off on how to accurately model the curvature of the Earth without suffering a significant performance penalty. For small geographic areas, a Cartesian coordinate system based on a plane tangent to the earth's surface will suffice. The resultant error is relatively small and typically acceptable for training. The error increases the further one travels from the tangent point, and so becomes impractical when the area represented gets too large.

For performance reasons, a flat earth representation has several advantages over a round earth representation. On a flat earth, the same direction always represents up. Line of sight calculations benefit from this by projecting the ray onto the terrain surface and testing height values at intersections to determine whether or not the terrain blocks the ray in 3D. Also, since the curvature of the earth is not taken into consideration, algorithms can take advantage of simple culling techniques to quickly eliminate a ray that is completely above a terrain area. In a round earth model, the calculations become much more complex.

Training simulators commonly use geodetic or geocentric coordinates to represent round earth terrain databases. Geodetic coordinates provide efficient point location lookup since each latitude-longitude pair maps to a unique location on the surface of the earth. Unfortunately, many calculations required by SAF behaviors, such as Euclidean distance between two points, are computationally expensive using geodetic coordinates. The reverse is true in geocentric coordinates. For this reason, many simulators, such as the OneSAF Objective System and WARSIM, use both.

However, there is a sizeable storage cost associated with storing the terrain in a round earth representation. The round earth model requires double-precision floats (64 bits) to achieve the precision required to perform accurate operations on the earth's surface using a geocentric or geodetic coordinate system. This means each coordinate requires a minimum of 24 bytes of storage; 48 bytes for both representations. For very dense or complex terrain, this can add up very quickly.

### Data Resolution

At the period of design for most current terrain representations, coarse data at 100 meter post spacing was the best widely available source data. While the NGA had defined higher resolution levels of digital terrain source data (DTED), they had limited availability and coverage. Fortunately, virtual and constructive systems usually do not have as great of a need for such high fidelity terrain data.

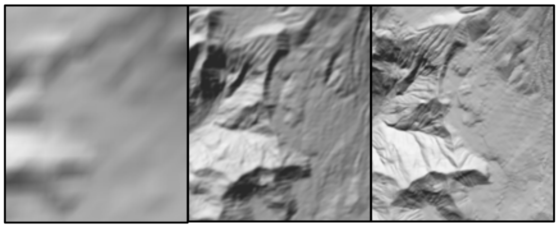

**Figure 4. A visual comparison of 100m (left), 30m (middle), and 10m (right) elevation data.**

However, live trainers such as OneTESS need much higher fidelity source than 100 meter data, to correlate to the real world. Fortunately, high resolution data is more readily available today. LIDAR data collection that supplies imagery and elevation data at meter or sub-meter resolution is becoming more commonplace. This data often overwhelms current systems and formats. To use such high-fidelity data, current terrain formats generally must perform considerable down sampling during the TIN generation process to meet their memory and storage requirements.

## THE LTF TERRAIN FORMAT

The typical terrain representations have various strong and weak points. Programs have specialized each of these to meet virtual and constructive simulator requirements. The unique OneTESS requirements demand a new solution. The live terrain format (LTF) was designed from the ground up to meet the specific needs of the OneTESS system, and the live training community in general. The design for the LTF borrows heavily from current industry standard terrain representations while incorporating design principles from other external industries, such as computer graphics and gaming. The major design goals for LTF are:

- A layered/scalable solution
- Small memory/storage footprint
- Optimized LOS performance
- Dynamic environment

### Layered/Scalable Solution

Many of the current synthetic environments store all of the terrain data together in one encompassing terrain data format. When executing terrain reasoning methods on the environment, all of the data is processed together. For instance, in OOS, the terrain features are integrated into the TIN. In this model, the terrain triangles and features are stored together and related. When a line of sight query occurs, a single algorithm processes both the terrain traversal and feature intersection check.

This “one size fits all” solution prohibits creating algorithms optimized specifically for a certain data type. In LTF, disparate data types are separated into their own storage and functional layers. This approach allows the development of specialized algorithms for each data type.

The layered architecture allows for a scalable and composable system. Currently, LTF is composed of the terrain and volumetric feature layers. In the future, developers could add new layers to enhance the system. For example, a system may require road networks to support automated route planning. Storing the road network in its own layer allows for the design of a specific, optimized route planning solution.

The layered architecture also enables users to compose their own system. In the above road network example, an individual live trainee has no use for the routing layer. The layered format will allow the trainee to configure their system to execute without it. Omitting the unneeded road network layer reduces the storage required on the resource-limited player unit.

### Terrain Pages/Round Earth

LTF stores the terrain data in pages that are one square kilometer in size. Each page has its own local tangent plane (LTP) coordinate system. The small size of the page minimizes the effects of earth curvature to within the 0.1 meter elevation error tolerance. Also the small page size allows the use of 32-bit floating point coordinates, which significantly reduces the memory requirements of the system.

A line of sight ray that overlaps multiple database pages is transformed into each page’s coordinate space before checking for blockage in that page. Since each page has its own coordinate system, the database can fully represent a round earth.

### Terrain Elevation Layer

To meet database size and performance requirements, the regular grid terrain surface representation has proven to be the best solution. The small 1-km$^2$ terrain page size and the “flat earth, 'Z' vector is always up” representation in each page gives suitable accuracy with regards to curvature of the earth while providing major storage space and runtime computation reductions compared to a geodetic or geocentric representation. The terrain surface is stored in a one-meter spaced grid, with each post representing the terrain surface height in decimeters as a 16-bit integer. The elevation grid for a single page only occupies 2,000,000 bytes of memory - roughly 2 MB.

In order to optimize the line-of-sight processing, the terrain is stored in a hierarchical tree. The terrain tree is composed of culling grids which contain the highest elevation of a 10 post by 10 post area. The LTF terrain skin, in its default configuration, is represented by a three-level tree. The lowest level is the 1-meter spaced grid, the middle grid is a 100x100 cell culling grid, and the top level is a 10x10 cell culling grid. The number of culling grids and the post spacing of the grids are configurable by the user.

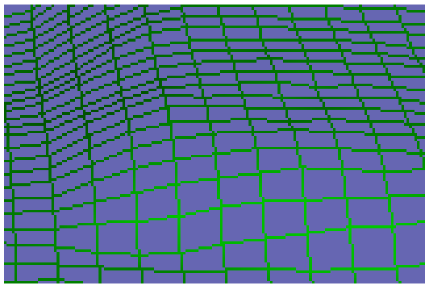
**Figure 5. Screen capture of 1 meter post spaced LTF terrain.**

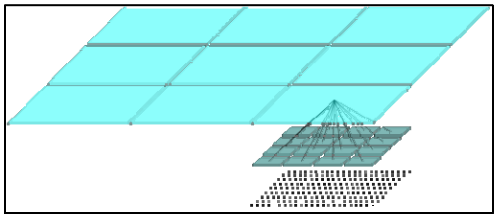
**Figure 6. Illustration of the culling grid hierarchy.**

The culling grids add minimal storage overhead to the tree structure. Table 1, shows the default configuration of the culling grids and the amount of additional memory required. The performance gain well justifies the slight increase in storage.

**Table 1. Memory requirements for a 1k x 1k elevation grid hierarchy.**

| Post Spacing (meters) | Number Posts | Post Size (bytes) | Total Size (bytes) |
|---|---|---|---|
| 1 | 1,000,000 | 2 | 2,000,000 |
| 10 | 10,000 | 2 | 20,000 |
| 100 | 100 | 2 | 200 |
| Total | | | 2,020,200 |

**Terrain Elevation LOS Method**

The field of computer graphics has studied algorithms to deal with two-dimensional grids for decades – after all, the computer monitor is basically a two-dimensional grid of pixels. All shape primitives drawn on the screen must be approximated as pixels. Bresenham's line drawing algorithm was created to approximate a line on the monitor. It determines the start and stop locations of a line, and finds all of the pixels that must be drawn. It has been highly optimized through the decades to make it as fast as possible.

The act of conducting a line-of-sight on a terrain grid is very similar to the problem that Bresenham solved. Basically, the LOS routine must know all of the grid "pixels" that the LOS ray overlaps, and check to see if the terrain in that grid blocks the ray. Unfortunately, since Bresenham's algorithm was created to approximate a line on the screen, it skips some pixels that the line overlaps to make the line appear straight. .

The 2DDDA (two-dimensional digital difference analyzer) algorithm is a derivative of Bresenham's algorithm. The major difference between it and Bresenham's is that it identifies every pixel that the line overlaps. The algorithm is highly optimized and traverses a regularly spaced grid quickly.

The LOS routine works recursively on the terrain tree. The routine starts on the top node of the tree and determines the start cell. The ray is checked against the height of the current cell. If the ray is lower than the height of that cell, the routine is called again on the child node of that cell. Otherwise, the routine traverses the grid to find the next cell in the current node to check. This continues until the end of the line is reached or the ray is blocked.

If a ray is close enough to the terrain skin, eventually it will work its way through the culling grids to the post grid. Each post in the post grid represents the elevation on a point in the terrain. The exact height at a location between the posts must be interpolated. Each cell of the post grid is conceptually represented by two right triangles. These triangles are created by drawing a line from the upper-left corner of a four post square to the lower right hand corner. When a ray needs be checked against the terrain skin, it is checked against the two terrain triangles. If the ray intersects either of the triangles, then the LOS routine is blocked. If not, the LOS traversal continues.

**Feature Representation**

Features include any objects other than the terrain that are capable of blocking line-of-sight. They are typically small in spatial size and quite numerous, which creates a significant challenge in using them efficiently. Examples of common features include trees, buildings, and street lights.

Virtual and constructive terrain database formats often take shortcuts in representing features – for example, OTF databases often represent entire forests as a single feature and use an attenuation model to simulate the probability of hitting individual trees. These kinds of shortcuts are not available in the live domain, which must represent every feature explicitly. In OneTESS, a feature is a solid object capable of blocking line-of-sight that is composed of a single material type.

LTF represents features as leaf nodes in a bounding-volume hierarchy (BVH) tree, which is a spatial tree commonly used by graphics and gaming applications for ray-tracing and collision detection. Each node in a BVH tree is a spatial volume that fully contains all of the volumes of its child nodes. Intermediate "culling" nodes organize features that are spatially close to each other so that a quick intersection check on the culling node can potentially eliminate a large number of feature nodes from consideration in the line-of-sight algorithm. BVH trees are somewhat slower for line-of-sight calculations than spatial-partition trees such as *k*D-trees, but were chosen for their significantly better update performance to support dynamic terrain events.

The LTF BVH tree supports using arbitrary geometry types for both intermediate "culling" nodes and feature geometry nodes through a common geometry interface. Any geometry that implements the requirements of the interface can be used, enabling rapid development of new geometry types for features. In the LTF prototype, only rectangular prisms and elliptical cylinders were implemented; a production version would also support ellipsoids, triangle meshes, and other common geometry types. When possible, features should be represented using simple solid geometry primitives instead of full triangle meshes since they enable faster computations and reduce memory use. Culling nodes should only use very simple solid geometry types like rectangular prisms since their intersection checks must be as fast as possible. Both leaf and culling nodes support full three-dimensional rotation using quaternions; features are not required to be oriented along the local z-axis.

Like other database formats, LTF supports the concept of attribution for features: any feature in the BVH tree can have a set of associated attributes. The only attribute currently defined for OneTESS is the "Material" attribute, which specifies the material type for the feature. Many properties that are commonly represented as attributes in other formats, such as feature length, width, and height, are instead explicitly represented by the feature's geometry in LTF.

**Feature LOS Algorithm**

The BVH tree used by LTF is optimized for attenuated line-of-sight computations. The algorithm is reentrant, and individual BVH tree nodes can be processed without requiring that the entire algorithm be run at once. The algorithm's state consists of a list of candidate BVH nodes that are known to intersect the line-of-sight ray, sorted by the distance of the intersection points from the ray's origin.

The algorithm starts by checking to see if the root node in the BVH tree intersects the line-of-sight ray. If so, it adds the root node to the candidate node list. After that, each time the algorithm is called, it checks and removes the next node in the candidate node list. If that node is a culling node, it performs an intersection query on each child of that node, and adds the child nodes that intersect the ray into the candidate node list. If the node is a leaf feature node, it provides the feature intersection to the attenuation model and asks if the feature completely blocks the ray; if so, it reports the blockage to the caller. The algorithm terminates when a blockage is detected or when no more nodes are left in the candidate node list.

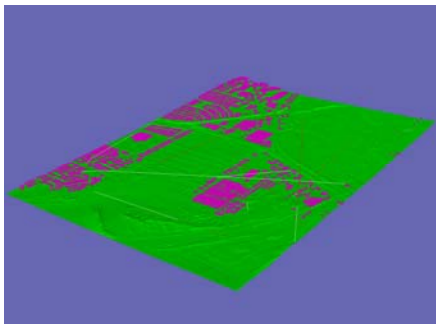

**Figure 7. Screen capture of LTF running a LOS against both terrain and features.**

The algorithm makes no assumptions about the attenuation model being used – attenuation is calculated through a generic interface that makes it possible to use completely different models for different runs of the algorithm. In OneTESS, the attenuation models will depend on the weapon and ammunition used for the bullet trajectory that the line-of-sight query represents.

**Feature Layer Construction**

The most difficult – and most crucial – part of constructing the feature layer is creating a good

BVH tree. The layout of the tree is the largest factor in the performance of the line-of-sight algorithm, since it determines how many nodes must be traversed for a query.

There are many heuristics used in BVH tree construction. The most challenging part is choosing the proper parent node for any particular leaf feature node. The goal is to group together features that are spatially close, and to minimize overlaps of sibling culling nodes. Overlaps destroy the culling efficiency of the tree since they often require two or more deep tree traversals to conduct checks on nodes that should have been grouped together.

The naïve approach is to only consider culling nodes that do not need to be enlarged to contain the inserted feature node, but this approach led to extremely large numbers of overlaps on the data sets. While culling nodes that already contain the feature should have priority over culling nodes that require enlargement, there are many heuristics to choose good parents when enlargement is required. Some of the more common include choosing the parent node that would require the least volume increase to use, choosing the parent node that would require the least surface area increase to use, and choosing the parent node that is spatially closest to the feature. LTF currently uses the minimum-volume-increase heuristic, but more research must be conducted to determine whether this is the best approach.

Other questions involve when to subdivide culling nodes that have too many children, and when to create new culling nodes rather than reuse existing ones. Currently, LTF caps each culling node to four children, subdividing if a fifth child is added. It also chooses to create new culling nodes at the root level if the best candidate parent's volume increase is above a threshold. These choices are fairly arbitrary, and should be studied in more detail to determine what the proper decisions should be.

**Dynamic Terrain Representation**

A live training system needs to be able to represent changes in the real environment that may occur during the exercise. These changes could include cratering, destruction of terrain features, and the creation of man made fortifications, such as berms, ditches and foxholes. These changes can potentially affect all of the LTF layers. For instance, a tree being knocked over would be feature change, while a crater from a shell would affect the terrain skin.

However, the current representation of the grid using one meter post spaced grids does not allow the needed resolution to properly integrate these terrain changes. A foxhole can be dug into the terrain in any location, not just on post boundaries. As stated previously, it is paramount to correlate with the dynamic events in the real world. The terrain grid has been designed to allow higher resolution grids in small areas. However, due to the size constraints of the player unit, this currently is not feasible. Therefore, all of the current dynamic terrain functionality is modeled with terrain features.

As previously discussed, the BVH structure is designed to be used with dynamic changes. The performance costs to create, modify, or delete a feature are less for it than with other tree structures. When a new feature is created, the correct bounding volume is identified and the feature is inserted. If needed, the containing volume will either have to be modified or created. If a feature is deleted or modified very minimal changes need to occur.

Since the terrain skin cannot handle dynamic events, all dynamic changes that alter the terrain skin must be modeled with features. Subtractive features are a special feature type which represents a volume where there is no terrain. Basically, when conducting an LOS query, if a ray intersects the terrain skin which is contained in a subtractive feature, then the ray is considered not blocked and continues. The subtractive features are a design for the future and are not currently implemented.

**Terrain Manager**

Even though OneTESS training will take place over relatively small geographic areas, the OneTESS player unit cannot accommodate the entire database in memory. The LTF runtime must be capable of efficiently swapping terrain tiles in and out of memory, caching them in a way that minimizes the number of swaps. OneTESS intends to keep at least 16 km$^2$ of terrain in memory, derived through consideration of typical line of sight distances and the memory constraints of the player unit.

The LTF Terrain Manager is responsible for properly loading tiles into memory when needed and determining which tiles to discard if too many tiles have been loaded. To do so, it uses a simple "Least Recently Used" (LRU) cache with a hard limit on memory use. This simple caching strategy should be sufficient for dismounts since their weaponry has a relative short range, they move relatively slowly, and most line-of-sight queries they conduct would occur within a 2 km radius of their location.

The Terrain Manager also acts as the top-level manager for line-of-sight queries: it determines which terrain tiles might intersect the line-of-sight ray and forwards the query to each tile in the proper sequence. The tiles are responsible for converting the ray into their local coordinate system and all further processing, and report whether the ray was clear or blocked in that tile.

## PRELIMINARY RESULTS

The prototype's results have been very encouraging. We developed a series of automated performance tests to simulate the expected types of line-of-sight queries in the OneTESS environment, and profiled the memory and disk use of the LTF runtime. All tests were conducted on the same workstation with the following specs:

| | |
|---|---|
| **CPU** | Intel Pentium D EM64T |
| **CPU Speed** | 3.00 Ghz (HT on) |
| **RAM** | 2 GB DDR |
| **OS** | Kubuntu Linux 7.04 (i386) |

Two 9 km$^2$ terrain databases were produced for testing purposes, both from Barstow, California. The grids were set to one meter post spacing, with 10 and 100 meter culling grids. A rectangular prism was generated for each building, and an elliptical prism was generated for each individual tree.

The Representative Terrain covers an area similar to the one expected for OneTESS Build 0 requirements, and has 1,255 volumetric features.

The Dense Terrain covers a section of downtown Barstow and is representative of a low-to-moderately dense urban area, with 8,714 volumetric features.

### Storage Benchmarks

**Table 2- Uncompressed Disk Size**

| Terrain DB | Grid (MB) | Features (KB) |
|---|---|---|
| Representative | 18 | 104 |
| Dense | 18 | 808 |

**Table 3- Compressed Disk Size**

| Terrain DB | Grid (MB) | Features (KB) |
|---|---|---|
| Representative | 1.7 | 104 |
| Dense | 1.7 | 304 |

**Table 4- In-Memory Size**

| Terrain DB | Grid (MB) | Features (KB) |
|---|---|---|
| Representative | 18 | 182 |
| Dense | 18 | 1,290 |

### Line of Sight Performance Benchmarks

**Table 5-LOS Query Times**

| Query Ray Length | Actual Distance Traveled | Rep. Terrain Query Time | Dense Terrain Query Time |
|---|---|---|---|
| 150 m | 76.8 m | 12.5 µs | 67.8 µs |
| 300 m | 121 m | 13.2 µs | 70.9 µs |
| 500 m | 160 m | 14.6 µs | 81.5 µs |
| 1000 m | 235 m | 16.9 µs | 83.9 µs |
| 1800 m | 348 m | 18.0 µs | 104.9 µs |
| 2000 m | 375 m | 18.4 µs | 115.1 µs |

**Table 6 – Linear Fit for LOS Query Times**

| Terrain DB | Intercept (µs) | Slope (µs / m) | Correlation |
|---|---|---|---|
| Rep. | 11.2 | 0.0148 | 0.985 |
| Dense | 54.3 | 0.151 | 0.960 |

The linear curve fit shows that, in these databases, the cost of a line-of-sight query is directly proportional to the length of the section of the ray processed by the algorithm. The intercept values show the overhead required on every query, and the slope value shows the processing cost per meter.

The lengths of the query ray were selected based on an analysis of weapon ranges for OneTESS. The actual distances traveled by the algorithm are less due to blockage. The provided actual distances come from the Dense Terrain; the corresponding distances for the Representative Terrain are higher since the ray is less likely to be blocked.

10,000 different rays of each query distance were randomly generated for the terrain, and each ray was queried 1,000 times. All query rays started and ended within 3 meters of the terrain surface.

## ADAPTING TO CONSTRUCTIVE / VIRTUAL DOMAINS

Virtual and constructive simulations can benefit from LTF for certain exercises where high fidelity environments are needed. The detail and performance are particularly suitable for urban operations. The accuracy LTF provides can capture ground truth specifics where source data is available to provide accurate detail needed for mission rehearsal.

To support virtual and constructive simulations, LTF would need extensions to include data not needed for live training. Examples include earth surface characteristics (soil type, vegetation coverage) for mobility, material composition for weapons effects and sensors, road and river networks for navigation, etc. The LTF design naturally supports these types of extensions as additional layers. This reinforces the underlying principle of optimizing the representation and algorithms for each layer of source data.

LTF could represent lower resolution terrain areas for virtual and constructive simulation. The gridded nature of the terrain surface is not locked into any specific spacing. Thus, the LTF accommodates 10-meter or other uniform spacing without modifications to the format or runtime services. Some simulations, such as CCTT, require reasonable slopes roads and flat lake surfaces. The LTF is designed to be able to support higher-resolution nested grids within the larger post spacing to capture additional detail in a similar manner to CCTT's cut-and-fill, substituting elevation posts for triangles. Simulations would realize the benefit of higher accuracy and greater run-time performance while using less storage.

## CONCLUSION

Live training in general, and OneTESS specifically, have the need for a high resolution environment to be able to correlate with the real world. Also, OneTESS has to operate on a handheld player unit with limited resources and processing power compared to desktop systems. These requirements are nothing like the current requirements for most virtual and constructive domains. Because of this there is no current format capable of meeting OneTESS's needs.

A new environment had to be created to meet the needs of OneTESS. This environment was built on common terrain representations along with technology from the graphics and gaming industries. LTF is a composable, scalable system capable of processing line-of-sight quickly on high resolution data. It has met its requirements in regards to both performance and resource utilization. And although it was designed specifically for the live domain, it can be expanded in the future to meet the needs of the virtual and constructive communities.

## References

Bresenham, J (1965) *Algorithm for Computer Control of a Digital Plotter*, IBM Systems Journal, 4(1):25-30

Lauterbach, Yoon, Tuft, Manocha (2006) RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs, *In Proc. 2006 IEEE Symposium on Interactive Ray Tracing*

Hall, R.J. & Chludzinski, J (2007) Timely Provision of Terrain Elevation Data for Geometric Pairing Computations. *In Proc. 2007 Intersercice/Industry Training, Simulation, and Education Conf.*

OneTess Program, *Synthetic Natural Environment OOS Reuse Report*, 29 December 2006