Unearthing the Modeling and Simulation Underground with Voxels

Freddie Santiago, John Moran, Jon Watkins
Dignitas Technologies
Orlando, Fl.
fsantiago@dignitastech.com, jmoran@dignitastech.com, jwatkins@dignitastech.com

Julio de la Cruz
ARL HRED ATSD
Orlando, Fl.
julio.a.delacruz4.civ@mail.mil

ABSTRACT

Adequate representation of underground structures is necessary for modeling and simulation (M&S) applications to provide quality training for both military and civilian use cases. Urban and asymmetric warfare is becoming more predominant, and training applications require technological advances in order to replicate the complexities of these urban environments. Underground roadways, basements, subways, sewer systems, and even subsurface multilevel buildings or garages are examples of the underground elements that play a role in urban and asymmetric warfare. Underground structures are important outside of urban environments as well. Naturally occurring underground features, such as cave systems, present special case challenges to existing synthetic environment algorithms that are tailored around open environments or structured building interiors. Given the prominence of aerial bombing to prepare for a ground invasion, complex military infrastructure will increasingly be found underground in bunkers or cave systems. In urban environments, infrastructure such as power, water, internet, and sewage, are already primarily underground, and require modeling to realistically replicate the urban battlespace. Simulations that include underground environments need to consider environmental effects, infrastructure interruptions, propagation of smoke and sound, fire propagation, and ventilation. This paper discusses an approach for representing and dynamically manipulating complex underground environments, both natural and man-made. The approach uses voxelization techniques found in modern game engines, movies, and medical imaging systems. The voxelized data for soil and structures is subdivided into chunks and organized into variations of B+ trees for efficient access and storage. Smooth-mesh algorithms analyze the voxel data and generate meshes for rendering and navigation mesh generation. We describe and propose an architecture for providing underground capabilities to existing and future simulation systems in a distributed fashion. Within this architecture, we also define functional components that will be modularized in order to take advantage of cloud computing technologies.

ABOUT THE AUTHORS

Mr. Freddie Santiago has over 10 years of applied experience related to modeling and simulation applications with a specific focus on terrain databases, synthetic environment services, dynamic terrain, and database correlation testing mechanisms for programs such as CDT, CCTT, AVCATT, UKCATT, SE Core A&I/CVE, OneSAF, and AGTS. He is presently the lead of research efforts into the representation of Underground Synthetic environments in military simulation for the Army Advanced Training and Simulation Division (ATSD).

Mr. John Moran has more than 9 years of experience designing and developing software. His focus has been on game design, game theory, and game engines. He is currently performing research to leverage gaming technology in military simulation and training.

Mr. Jon Watkins is the founder and Chief Operating Officer of Dignitas Technologies, LLC. He has 25 years of applied M&S experience, with a focus on geospatial databases, database generation, synthetic natural environment (SNE) services, dynamic environments, and entity-level movement control algorithms on systems such as SIMNET, ModSAF/OTB, CCTT, JSIMS/WARSIM, OneSAF, and Common Driver Trainer.

Julio de la Cruz is the Chief Engineer for Synthetic Environments Research at the Simulation and Training Technology Center in Orlando, FL and is responsible for the research and development of applied simulation technologies for learning, training, testing and mission rehearsal. He has a B.S. degree in Electrical Engineering from the City College of New York and a M.S. in Industrial Engineering from Texas A&M University. He is a graduate of the U.S. Army School of Engineering & Logistics and alumnus of the Advanced Program Managers Course at the Defense Systems Management College (DSMC).

Unearthing the Modeling and Simulation Underground with Voxels

Freddie Santiago, John Moran, Jon Watkins
Dignitas Technologies
Orlando, Fl.
fsantiago@dignitastech.com, jmoran@dignitastech.com, jwatkins@dignitastech.com

Julio de la Cruz
ARL HRED ATSD
Orlando, Fl.
julio.a.delacruz4.civ@mail.mil

OVERVIEW

Modeling and Simulation (M&S) applications must represent environments and situations which are inherently unusual or difficult to train live, such as within urban or underground environments. While early first person shooter games (e.g. Doom, Quake) started in dungeons, most early M&S applications (SIMNET, CCTT, etc.) addressed the outdoor case first to support the most common training environments. However, this has left a significant shortfall in the ability of the M&S community to represent, operate on, and thus train within underground environments. As urban and asymmetric warfare become more predominant, including potential conflicts within megacities, training applications will require technological advances in order to replicate the complexities of dense urban and underground environments. Examples of such environments include underground roadways, basements, subways, sewer systems, and even subsurface multilevel buildings and garages.

Underground structures are also important outside of urban environments. Examples include Osama Bin Laden's suspected use of cave complexes in Afghanistan, North Korean underground complexes, and the Viet Cong's extensive use of tunnel networks in Vietnam. In conventional warfare, underground bunkers and communication lines are critical to military command and control. Natural features, such as cave systems, also present special case challenges to synthetic environment algorithms that are tailored around either open environments or very structured building interiors.

This paper focuses on research done for ARL HRDE ATSD to solve the problem of representing those environments in modern M&S systems. We will discuss analysis performed on existing technologies, outline an architecture, describe prototypes, and discuss the performance metrics captured.

PROBLEM STATEMENT

Underground environments are difficult to represent because structures and atmosphere in such environments behave in a unique manner, which is different from above ground. The simple fact that the structure is below ground completely throws off models currently used for synthetic environments. Any damage to an underground structure may result in a catastrophic collapse, which is hard to model in current systems. Many other potential hazards, like fire, are significantly more problematic than in above-surface environments. Above-ground simulations often include fires, which are can be ignored as scene clutter because they are not necessarily threatening. However, an uncontrolled fire in an underground environment is potentially catastrophic. Lack of oxygen, smoke build up, and heat all require different operational choices and could cause missions to fail or be significantly altered.

While the underground representation problem spans multiple challenging topics, we focus on the problem of subsurface structural damage and collapse, which is a large hurdle for current M&S systems. Our goal is to bridge the gap between current systems and future technology in these environments.

ANALYSIS

Our initial research goal was to define a mechanism for damaging an underground structure, then causing it to collapse. But, if we break a wall and cause it to collapse, what is beyond the wall? We must represent the volume of terrain that envelops any underground structure. This is different than most above-surface situations since in those cases the terrain could be represented as a 2-dimensional “blanket” of triangles that undulates to mimic the rise and fall of the terrain. This works for above-surface simulations since all entities are expected to remain above the terrain surface. In the underground, however, we need to represent the composition of the soil at any level on, or below, the terrain surface.

Previous solutions implemented mechanisms for representing the subsurface soil by combining the use of soil posts and height maps (Dukstein G. 2013). The soil posts contain the composition of the soil at a specific depth from the soil surface. The height maps define the terrain surface as the dynamic events happen (e.g. digging). This approach is ideal for situations where the simulation only needs to dig straight down. However, they fall short of the capability necessary to simulate horizontal digging, as in tunnels or mines.

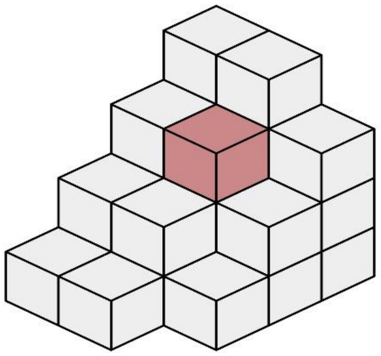


Figure 1: Voxel Representation. Single voxel is shaded

Other, more traditional, approaches handle underground structures by generating “fixed” models that are embedded in the terrain. This approach allows entities to maneuver under the terrain surface. However, the nature of the models does not typically lend itself well to dynamic modifications. Thus, the use case of collapsible structures remains out of reach, or has to be kludged. These fixed models also do not represent natural underground structures well, such as caves, since the models often have very rigid components and restrictions.

Games like Minecraft have experienced success representing the terrain using voxel-like structures. A voxel is a data structure used to contain a value, or volume, on a regular grid in three-dimensional space (See Figure 1). Given this, we investigated representing the subsoil in a similar fashion as Minecraft. Could we represent the terrain as voxels, and could we somehow create smooth meshes to make it look realistic? The Minecraft worlds, although excellent at representing very coarse terrain volumes, would not provide a high resolution view suitable for immersive training due to their blocky representation (See Figure 2).

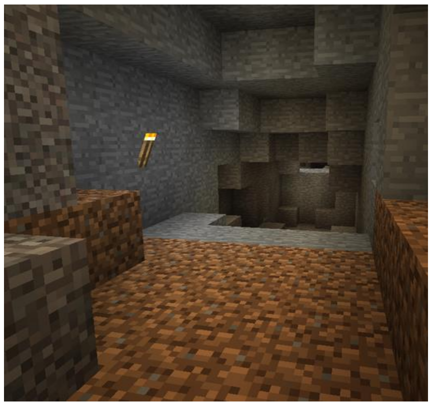


Figure 2: Minecraft Voxel Rendering

In order to identify technology gaps with the voxel approach, we experimented with generating voxels and wrapping them around structures. This is necessary in the case where a wall is breached and a simulation system needs to represent the soils beyond the wall. We used Unreal Engine 4 to develop a prototype that procedurally generates boxes (i.e. voxels) around a given structure.

We experimented with wrapping voxels around structures to identify any other technology gaps that should be considered. Our experiments focused on procedurally generating boxes (i.e. voxels) around a given model to see how tightly we could wrap the voxels around the model while still maintaining a rendering rate of approximately 30 frames per second (fps). We found that we could maintain 30fps rendering rate using approximately 8000 static boxes around a cylinder (i.e. mock tunnel) in a 1km cube area.

Using our initial results as benchmarks, we searched for software libraries that specialize in voxelization to see if we could improve the performance. We required that libraries support operations on the voxel data, such as intersection, subtraction, and addition. We analyzed a few libraries and engines before selecting OpenVDB because of its efficient storage and handling of voxel data, its support of dynamic operation on the data, the active community of users and developers, and its open (free) license.

OpenVDB

After some initial research and experimentation with several voxel technologies and approaches, we selected the OpenVDB library to use for our prototype. OpenVDB is so named because it is an open source library that uses the

VDB datatype, a volumetric, dynamic grid that shares several characteristics with B+ trees (Museth, K 2013). OpenVDB is a C++ library for the manipulation of sparse, time-varying, volumetric data on three-dimensional grids. It is actively developed and maintained by Dreamworks Animation. Dreamworks has used OpenVDB for modeling fluids, clouds, and large scale terrain fracturing in movies (See Figure 3). At its core, OpenVDB handles data using sparse B+ trees. This method of storing and accessing voxels is beneficial to our use case of representing underground soil, as described below (DreamWorks Animation, About OpenVDB).

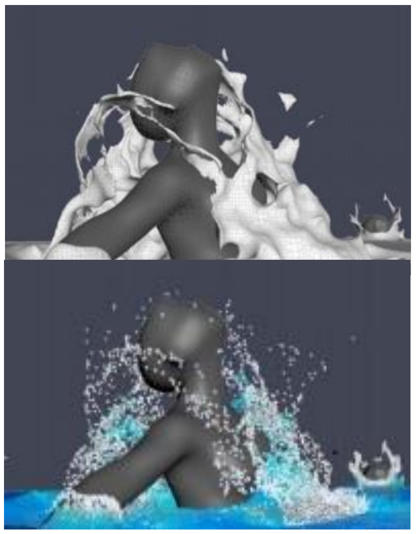


Figure 3: Fluid Dynamics with OpenVDB

Storage

Storing data from a sparse B+ saves disk space when compared to other methods, because only relevant data is stored. The sparse VDB B+ tree also allows for a potentially infinite growing space. There is no need to define a world space bounding box beforehand. This allows for "digging" indefinitely, as well as the wide variety of complexity found with underground structures, natural or artificial. OpenVDB includes methods of compressing data when doing file input/output (IO) which allows for efficient use of hard disk space. This compression is critical as the scale of M&S environment databases grows.

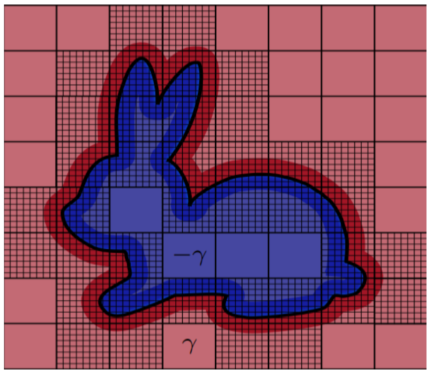


Figure 4: Narrow-band Level Set Example

Typelessness

Volumetric data describes points in space and attaches some value and meaning to each point. OpenVDB allows storage of any type of data at each point, to use in different ways. Primarily, we use floating point valued trees which represent renderable volumes. OpenVDB refers to these as Level Sets. Additionally, OpenVDB trees can store vector valued points, where each voxel represents a flow direction applied to fog, liquid, or particles.

Narrow-band Level Set

The narrow-band level set feature of OpenVDB distinguishes between "active" and "inactive" voxels, which limits the number of voxels in play at any time. Only the voxels which are within a certain distance to the surface are considered active and all other voxels are inactive, and set to the background value. The implication is that only the voxels which are close to the surface of the volume are stored in memory. All other voxel locations are assumed to be default background values.

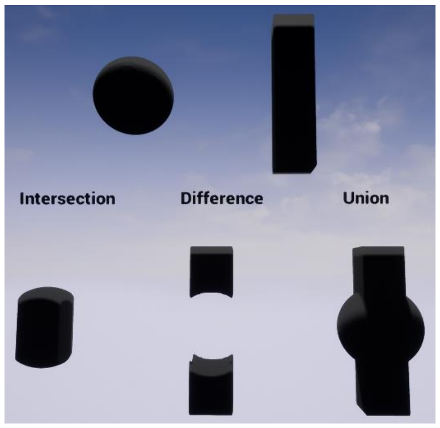


Figure 5: OpenVDB Geometry Operations

The image in Figure 4 shows a 2D slice of what a volume looks like in memory and how OpenVDB interprets narrow band level sets. Tiles (i.e. large areas of space) are marked with γ. These are background tiles and if a coordinate is accessed in the tile it returns the default background value. The red regions are outside the volume and contain positive values. The blue regions are inside the volume and contain negative values. The darkly shaded areas make up the narrow-band of active voxels.

Geometric Operations and Dynamic Terrain

OpenVDB provides algorithms for geometric operations on two voxel grids, including intersection, difference, and union. Each function takes in two grids and performs the respective operation (See Figure 5). We use this functionality to perform dynamic terrain updates at runtime as well as to dynamically generate the underground soil when given a terrain height map along with underground structures.

Mesh Generation

OpenVDB uses a specialized version of a ray-marching algorithm known as Hierarchical Digital Differential Analyzer (DDA) to create the topology of level sets (i.e. meshes) for rendering. The specialized version takes concepts of ray-marching DDAs and expands them to leverage the hierarchical structure used in OpenVDB (Museth, K. 2014).

UNDERGROUND PROTOTYPE SYSTEM

Our ultimate research goal is to design a system that represents underground environments, and can be used by current and future simulation systems. To achieve this, we created a prototype server to handle the special cases of underground data. In operation, a simulation system detects the need for special processing, then calls upon this server software to return the results. In this section, we provide an overview of our current design for this server software, dubbed “Cerberus”.

Architecture

The Cerberus architecture (see Figure 6) is composed of two communication pipelines. The first is the client-to-Cerberus communications pipeline. This pipeline handles all client requests for data and all responses from Cerberus. Cerberus provides a Dynamic Linked Library (DLL) that is linked into the simulation system, along with an Application Programming Interface (API) which defines all the available interfaces. Typical API functionality includes requests such as: Line of Sight, Path Planning, Handling of Dynamic Events (e.g. Detonations), Ground Elevation queries, etc. The Cerberus client DLL is responsible for handling the communications between the clients and the Cerberus gateway.

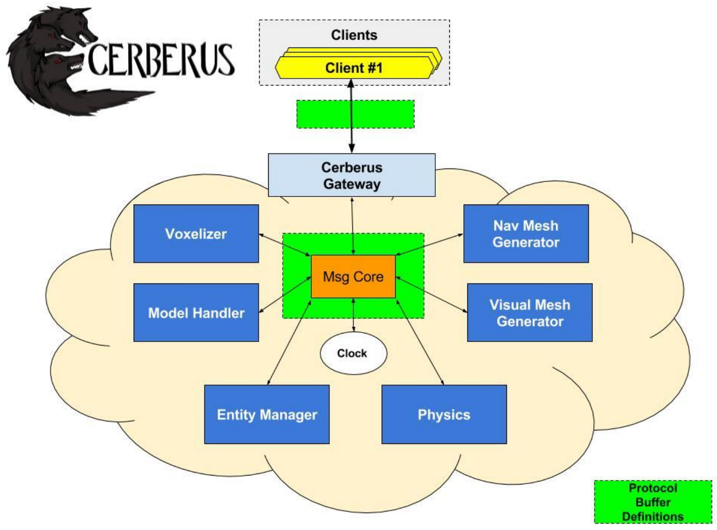


Figure 6. Cerberus High Level Architecture

The second pipeline handles the internal communication between all Cerberus system modules. Within Cerberus, modules contain major areas of functionality (i.e. Voxelizer, Physics, Navigation Mesh Generator, etc.). Modules communicate with each other and with the Gateway using a publisher-subscriber messaging mechanism. This way each module subscribes to specific topics that are important to the module. For example, the Physics module may care about the CerberusSystem.Clock exchange, or the CerberusSystem.Explosions exchange, but it may not necessarily care about a CerberusSystem.LightSource exchange. Using a publisher-subscriber protocol, such as RabbitMQ, also gives the Cerberus system flexibility to host all modules on one machine, or distribute the modules out to other system nodes (i.e. computers) to handle the processing.

Runtime Execution

The Cerberus idle state consists of the Gateway application running on one network node. Any available system node runs a Cerberus Service which listens for commands from the Gateway. These Cerberus services register with the Gateway to let it know the node is available to handle processing.

When a client (i.e. simulation system) is initializing a simulation scenario, it calls on the Gateway to load a specific dataset (e.g. Hawaii). Note this request is actually sent from the Cerberus client DLL that is integrated with the client’s simulation software. The Gateway receives the load command, along with any configuration commands, and begins initializing the Cerberus System. First, the Gateway determines the best system nodes to handle specific processing. For example, the Gateway may choose the node with the most processing power in the system to handle the Physics module. Upon selection, the Gateway sends a command to each of the system nodes to begin initializing the specific modules. One system node may handle more than one module (e.g. Navigation Mesh Generation and Visual Mesh Generation). Once each module has finished initializing it sends a message to the gateway to communicate its status. When all modules are online the Gateway responds to the client and communicates a *system ready* status. When the client sends a simulation start command, the Gateway commands all modules, including the Cerberus Clock, to begin. This clock is used to synchronize the operation of each module.

Once the initialization sequence is complete, the system is ready to handle any requests from the clients. Upon receiving a request from a client (e.g. Handle Explosion Event), the Gateway publishes the request to the appropriate exchanges. The initial implementation of the Cerberus prototype routes the request to each of the modules serially so that each module can publish its results and other modules can ingest those results and continue processing. However, the long term vision for the Cerberus software is to publish and process data as it arrives without the modules waiting on the “green light” from the Gateway to begin execution.

Voxelization Module using OpenVDB

OpenVDB provides a tool that allows us to accurately represent the subsoil in underground environments, through processing of existing terrain databases to generate voxel grids. This supports our target use case of supporting dynamic soil modifications at runtime. This use case is shown in Figure **7**7, where a Soldier is digging through terrain, reaching another bunker, and breaching the bunker.

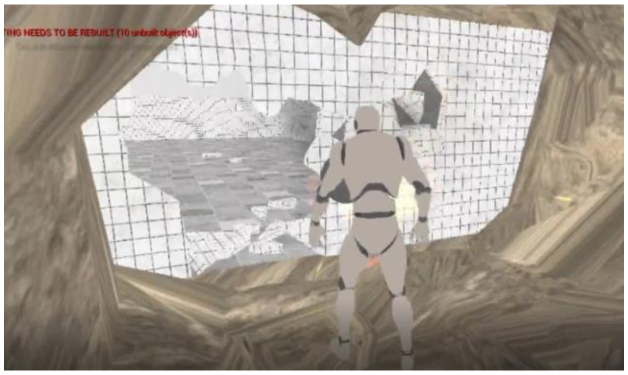

ING NEEDS TO BE REBUILT (10 unbuilt object(s))

Figure 7: Cerberus Digging Example

OpenVDB also allows us to create volumes from models and serve them to clients at different resolutions. This is particularly useful when taking cross platform services into account. We generate more detailed and accurate meshes at lower voxel sizes (i.e. higher resolution) for desktop users, and less detailed meshes at higher voxel sizes (i.e. lower resolution) for mobile users. OpenVDB is capable of generating different resolution grids from detailed grids via mip map levels. The higher mip levels provide less detail, but also fewer vertices and triangles (see Figure 8).

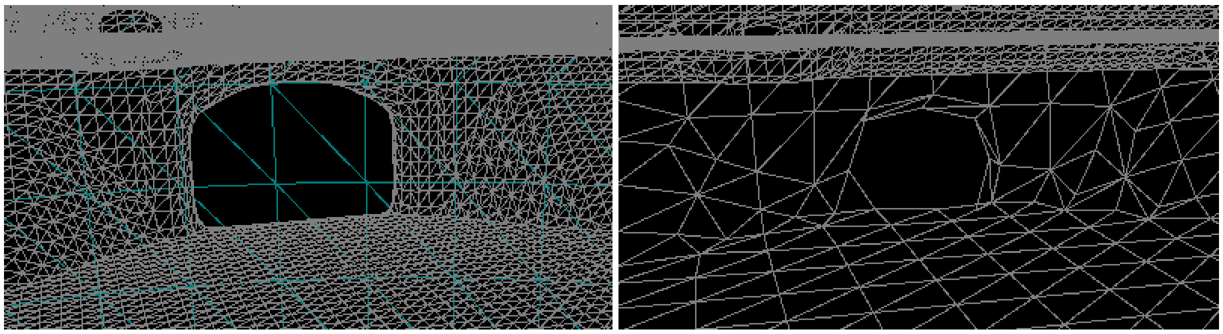


Figure 8: Voxel Generation of a Tunnel Entry. High Resolution on the left (Mip Level 0) , Low Resolution on the right (Mip Level 3)

Scalability

While OpenVDB is designed with performance and parallelization in mind, these concepts must scale to handle entire cities or regions. Additionally, by default the dirt volume is generated as a single mesh for the entire region, which means that dynamic updates, such as digging, need to update the entire render mesh and the entire collision mesh. We developed a VDB Grid Manager to solve this problem. The Grid Manager divides the volumes into smaller, equally-sized sub sections and manages them as a unit. Geometry functions are applied specifically to the affected subsections of the volume rather than a single monolithic volume. The client updates only affected sub sections and not the entire terrain mesh. This method also lends itself to mass parallelization as the processing for each individual subsection can be off loaded onto separate threads, or separate machines

PERFORMANCE EVALUATION AND TUNING

Keeping in mind our goal of updating the subsoil representation dynamically at runtime, we evaluated the performance of our prototype approach. OpenVDB has many parameters which influence how volumes are generated and changes to these values affect the efficiency of the voxel representation and the speed of modifications. We considered two aspects of voxelization in our evaluation. First, we looked at the preprocessing of databases for areas to be simulated. This includes voxelizing all models, generating the underground terrain, as well as breaking down the generated voxel data into discrete sections (i.e. pages). Then, we considered the at-runtime and in-real-time dynamic updating of voxelized models, which are needed for fracturing or digging.

The steps for the experiment are as follows:
1. Given a terrain mesh, extract all the vertices and polygons then pass those to the voxelization algorithm
2. Measure the time to create the volume from the given geometry (i.e. preprocessing time)
3. Render the generated volume
4. Shoot 4 balls at the mesh from a predefined location and direction
    a. When a ball collides with the mesh, a sphere volume is generated at the hit location, and a geometric difference (i.e. subtraction) between the terrain volume and the sphere is calculated
    b. Capture the time to update the voxel grids (i.e. update time)
5. Calculate the average of the update times and record that as the update time for the specified configuration

The parameters measured are:
- Voxel size: The size in world units of each voxel cell of the tree
- Adaptivity: A value from 0-1 which allows the meshing algorithm to generate a mesh with less vertices and triangles while also being less accurate. The higher the value, the simpler the mesh.
- Split Depth: The number of times the mesh was uniformly subdivided, into equally sized smaller volumes
- Mip Level: Determines the number of model simplifications (i.e. mip levels) that should be generated

We ran separate tests to measure each parameter and the implications of each. In general, a higher voxel size meant a lower resolution and therefore less total voxels. A decrease in both preprocessing time and update time was observed as voxel size increased.

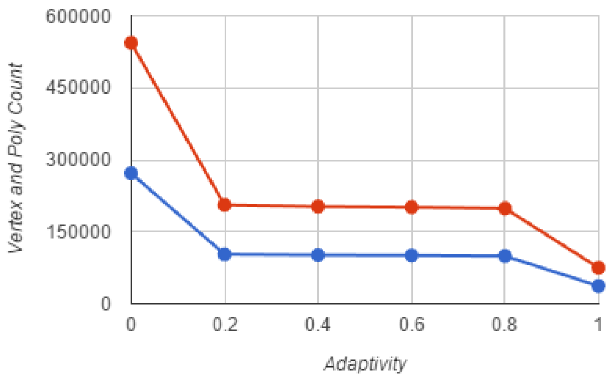
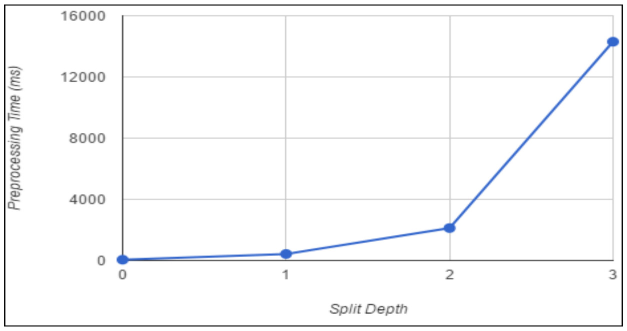


Figure 9: Adjustment in Adaptivity value and resulting vertex and polygon count

For adaptivity, we saw some negligible improvements in preprocessing time and update time as the adaptivity value was increased (see Figure **9**9). Although the adaptivity does not have a significant impact on the processing and update times it is expected to have a significant influence on the time it takes to render the volumes on the client side. The graph shows approximately a six times decrease in the polygon count and three times decrease in vertex count from adaptivity 0 to adaptivity 1. This means the resulting mesh isn't as accurate, but it may be acceptable in certain cases. One example might be to use the adaptivity 1 mesh for collision and the adaptivity 0 mesh for rendering.

The mip level parameter used for OpenVDB's multiresolution grids proved significant decreases in update times at the cost of preprocessing times. However, even at a mip level of 1, significant amounts of detail are lost very quickly in the resulting mesh, which may result in unusable meshes for rendering.



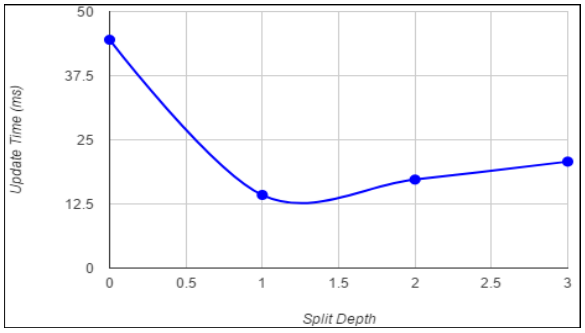
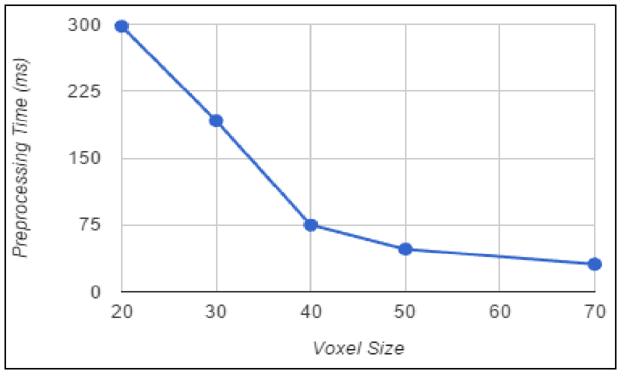


Figure 10: Split Depth parameter effects on Processing Time (left) and Update Time (right)



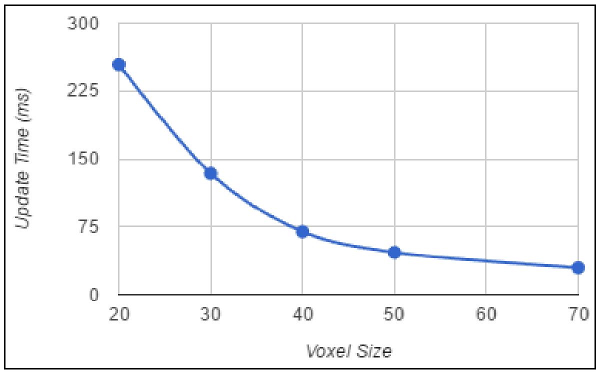


Figure 11: Voxel Size parameter effects on Processing Time (left) and Update Time (right)

As mentioned above, the split depth parameter is the number of times the volume was uniformly subdivided, via oct splitting, into equally sized smaller volumes. A split depth of 0 indicates no subdivisions. At higher split depths the number of subdivisions is equal to 8^X, where X is the split depth. The graphs above were captured using a voxel size of 50. A similar trend is held regardless of voxel size. In the voxel size graph at voxel size 50, update times took approximately 40ms per update (Figure 111). However, after splitting the volumes into sections, using a split depth of 1, the update time went down to 13ms. The increase in update times after a split depth of 1 (8 sub sections) is seen due to the collision checks required to see whether specific subdivisions were affected by the dynamic events (e.g. explosions). Similarly, there is also an increase to the preprocessing time to generate the volume as there is an added step to generating the volumes. Our tests only went as far as a split depth of 3 because of the exponential increase in preprocessing time. The model used for testing was a terrain piece that represented 100m^2 of space and a typical page in a database includes about 5,000m^2 of models. Given this, the exponential increase in preprocessing time required for higher split depths wouldn't be viable when scaled up. Additionally, as seen in the update graph, higher split depths followed an increasing trend, so there would be no added benefit.

These experiments were done without multithreading and on a single CPU. We proved that using the grid manager does reduce runtime processing time in a single thread environment, so now we have confidence to expand this concept to be multithreaded and parallelized using cloud technologies. Each individual subdivision can be off loaded and processed separately and result in different metrics, but this gives us a base starting point.

We continue to experiment with these metrics to find an appropriate balance. For example, the multiresolution grid can be useful if an initial volume has a small voxel size (i.e. high resolution) and the system needs to downscale it to provide a lower resolution volume to the client. As mentioned above, higher voxel sizes in addition to higher mip levels cause the resulting volume to be far less accurate to the original mesh, therefore, it might be detrimental to model integrity and immersion if a low resolution volume is used as a base for the multiresolution grid. A similar effect happens when going from very small voxel sizes to very large voxel sizes. There is not likely a one size fits all solution, since each simulation platform has different needs. Therefore, we are designing the system to allow clients to configure the system to suit their needs, or choose between a set of predetermined parameter values.

CONCLUSION

The experiments show that voxelization of subsoil is a viable solution for generating and manipulating underground environments. The control parameters available facilitate the tailoring of voxel generation to fit specific use cases, including adjustments for density and detail. Performance metrics show that updates to the voxel meshes fall within acceptable ranges for runtime simulation execution. Potential improvements could arise from parallelizing the processes and modules outlined. All these factors combined present a compelling case for the viability of the technology for military and civilian simulations.

FUTURE WORK

As the software experimentation and development continues to progress, additional elements of underground environments will be considered. Upcoming plans for our research, funded through ARL HRED ATSD, includes using voxels to provide structural integrity properties and fragmentation patterns; exploring atmospheric modeling using the vector field capabilities of OpenVDB; and considering fire, smoke, and air circulation, which are critical to underground environments. Outcomes of our research will contribute to the refinement of our prototype and bridge the gap of current underground simulation technology.

REFERENCES

1. Watkins, J.; Campbell, C. (April 2016) “Technical Challenges for Simulation and Training in Megacities” Small Wars Journal, Feb 2016 (http://smallwarsjournal.com/jrnl/art/technical-challenges-for-simulation-and-training-in-megacities). Presented at TRADOC G2 Mad Scientist Megacities Conference.
2. Dukstein, G., Watkins, J., Le, K., and Gonzalez, H., (December 2013) “Extending construction simulators through commonality and innovative research,” in Proceedings of the Interservice/Industry Training, Simulation, and Education Conference, pp. 2355–2365, Orlando, Fla, USA,
3. DreamWorks Animation. About OpenVDB [web page] Retrieved from http://www.openvdb.org/about/
4. Ju, T., Schaffer, S., and Warren, J. (2002) Dual Contouring of Hermite Data. Retrieved from http://www.frankpetterson.com/publications/dualcontour/dualcontour.pdf
5. Lorensen, W. E., and Cline, H. E. (1987) Marching Cubes: A High Resolution 3D Surface Construction Algorithm Retrieved from http://academy.cba.mit.edu/classes/scanning_printing/MarchingCubes.pdf
6. Museth, K., Lynch, J., Budsberg, J., and Bailey, D. (2015) OpenVDB Introduction. [presentation] Retrieved from http://www.openvdb.org/download/openvdb_introduction_2015.pdf
7. DreamWorks Animation (2012) OpenVDB at DWA [presentation] Retrieved from http://www.openvdb.org/download/openvdb_production_2015.pdf
8. Museth, K. (2013) VDB: High-resolution sparse volumes with dynamic topology. Retrieved from http://www.museth.org/Ken/Publications_files/Museth_TOG13.pdf
9. Museth, K. (2014) Hierarchical Digital Differential Analyzer for Efficient Ray-Marching in OpenVDB. Retrieved from http://www.museth.org/Ken/Publications_files/Museth_SIG14.pdf