ERDC/ITL SR-20-3

US Army Corps
of Engineers®
Engineer Research and
Development Center

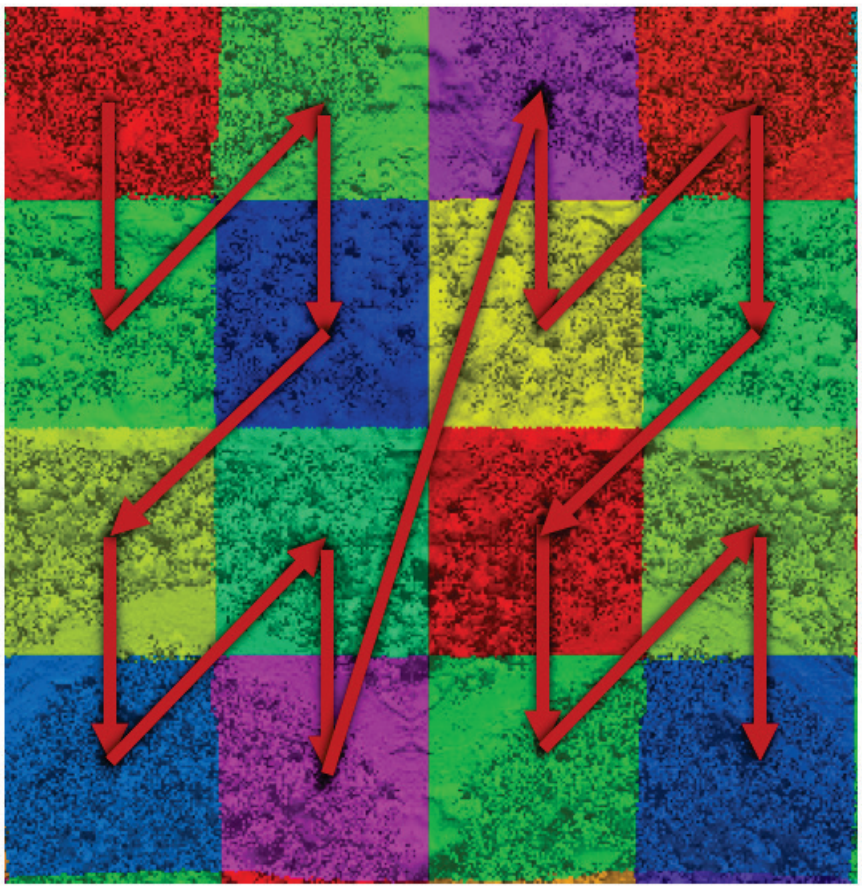
ERDC
ENGINEER RESEARCH & DEVELOPMENT CENTER

Using Morton Codes to Partition Faceted Geometry

An Architecture for Terabyte-Scale Geometry Models

Robert H. Hunter, Barry C. White, Reena R. Patel,
and Jerrell R. Ballard, Jr.

April 2020



Information Technology Laboratory

The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdc.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at http://acwc.sdp.sirsi.net/client/default.

ERDC/ITL SR-20-3
April 2020

Using Morton Codes to Partition Faceted Geometry

An Architecture for Terabyte-Scale Geometry Models

Robert H. Hunter, Barry C. White, Reena R. Patel, and Jerrell R. Ballard, Jr.

Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under MIPR Number HQ0642033036

Abstract

The Virtual Environment for Sensor Performance Assessment (VESPA) project requires enormous, high-fidelity landscape models to generate synthetic sensor imagery with little to no artificial artifacts. These high-fidelity landscapes require a memory footprint substantially larger than a single High Performance Computer's (HPC) compute node's local memory. Processing geometries this size requires distributing the geometry over multiple compute nodes instead of including a full copy in each compute node, the common approach in parallel modeling applications. To process these geometric models in parallel memory on a high-performance computing system, the Geometry Engine component of the VESPA project includes an architecture for partitioning the geometry spatially using Morton codes and MPI (Message Passing Interface) collective communication routines. The methods used for this partitioning process will be addressed in this report. Incorporating this distributed architecture into the Geometry Engine provides the capability to distribute and perform parallel ray casting on landscape geometries over a Terabyte in size. Test case timings demonstrate scalable speedups as the number of processes are increased on an HPC machine.

Contents

Figures and Tables

Figures

Tables

Preface

This study and report was authorized by U.S. Army Engineer Research and Development Center (ERDC), under the Engineered Resilient Systems (ERS) Program, Sensor Systems work package, MIPR Number HQ0642033036. Technical Director for ERS was Dr. Robert M. Wallace.

The work was performed by the Computational Analysis Branch (IE-C) of the Computational Science and Engineering Division (CSED), U.S. Army Engineer Research and Development Center, Information Technology Laboratory (ERDC-ITL). At the time of publication, Dr. Jeffrey Hensley was Chief, CEERD-IE-C; Dr. Jerrell R. Ballard, Jr. was Chief, CSED; and Dr. Robert M. Wallace, was the Technical Director for ERS. The Deputy Director of ERDC-ITL was Ms. Patti S. Duett and the Director was Dr. David A. Horner.

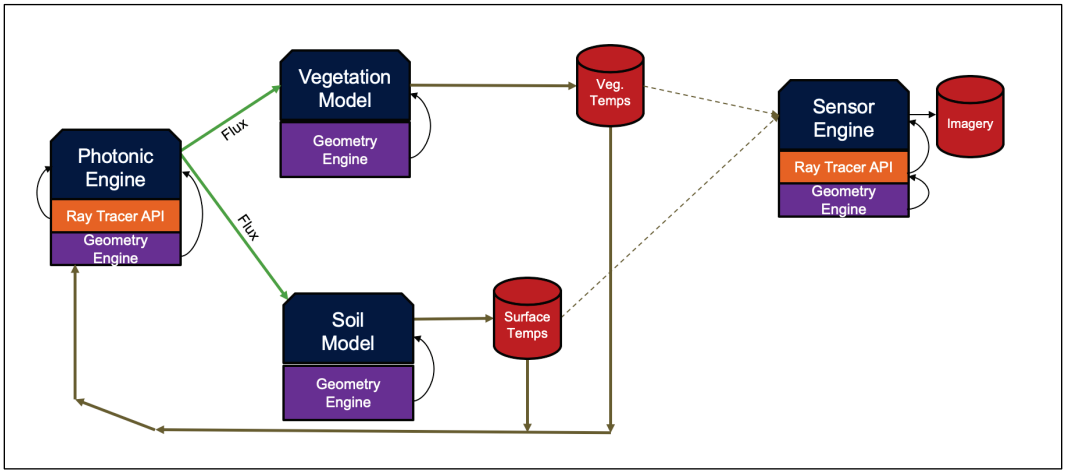
COL Teresa A. Schlosser was the Commander of ERDC, and Dr. David W. Pittman was the Director.

1 Introduction

1.1 Background

The goal of the Virtual Environment for Sensor Assessment (VESPA) project is to employ synthetic imaging sensor models to generate realistic infrared images of authentic landscapes. There are multiple software components of the project that go into producing this sensor simulation. Figure 1 provides an overview of these software components and their relationships to each other.

Figure 1. Overview of the VESPA simulation workflow.



The Geometry Engine component is responsible for loading the landscape models used by the simulation to provide radiative data to the Sensor Engine. In order to handle complex, memory-intensive scenes, the Geometry Engine distributes the landscape geometry across multiple compute nodes of a High Performance Computer (HPC) to perform ray casting in a parallel-memory environment. Landscape geometries processed by the Geometry Engine include a soil surface mesh and a vegetation mesh. Each of these meshes is a list of coordinate vertices and a list of triangular facets, following the polygonal-list based structure for three-dimensional objects described by Watt and Watt (1992). Facets have three ordered indices into the coordinate vertex list, defining a triangle in three-dimensional space. The method used to partition the landscape and vegetation geometry for this distribution is the topic of this report.

1.2 Objective

The Geometry Engine for the VESPA project is responsible for providing radiative energy emitted from surfaces to the Sensor Engine as realistically as possible, reducing the number of artificial artifacts. Thus, the engine must be able to handle high-fidelity landscape geometries. In addition, some of the images generated by the VESPA project could be perspectives from flyover sensors, which means the landscapes have to cover a wide area. Both requirements dictate that the Geometry Engine would need to handle landscape with vegetation geometries that are extremely large in terms of their memory footprint.

Previous parallel radiative transfer analysis on geometry models involved storing a full copy of the geometry in each HPC node of the parallel program. This limited the amount of communication needed between each HPC node, as each MPI (Message Passing Interface) process had all of the data stored in local memory and indexed computed values were all that needed to be transferred (Howington et al. 2019). However, this created a limitation on the size of the geometry that could be analyzed; the geometry must be able to fit in the memory of a single HPC node. The high-fidelity landscape models required by the VESPA project for realistic imagery could far exceed the size of an HPC compute node's local memory. Working with these landscapes required a different architecture that could distribute a geometry across multiple nodes and perform ray tracing in parallel on that distributed mesh. This report will discuss and provide results of the architecture used to accomplish this.

1.3 Approach

The algorithm used to partition a landscape geometry containing billions of triangular facets across multiple HPC nodes involves performing a parallel sort of the facets using a spatial distribution, grouping them by spatial locality. Morton codes are used as a one-dimensional spatial sorting key. A parallel sort is completed using MPI to redistribute facets to processes based on an assignment technique using the facet's Morton code. After the sort is complete, each process has a spatial chunk of the mesh, which can then be prepared for ray tracing. The following sections will describe this algorithm in more detail, specifically focusing on how Morton codes facilitate the spatial sorting step.

1.4 Terminology

Table 1 lists and defines the terms used throughout this report.

Table 1. Terminology used throughout this report.

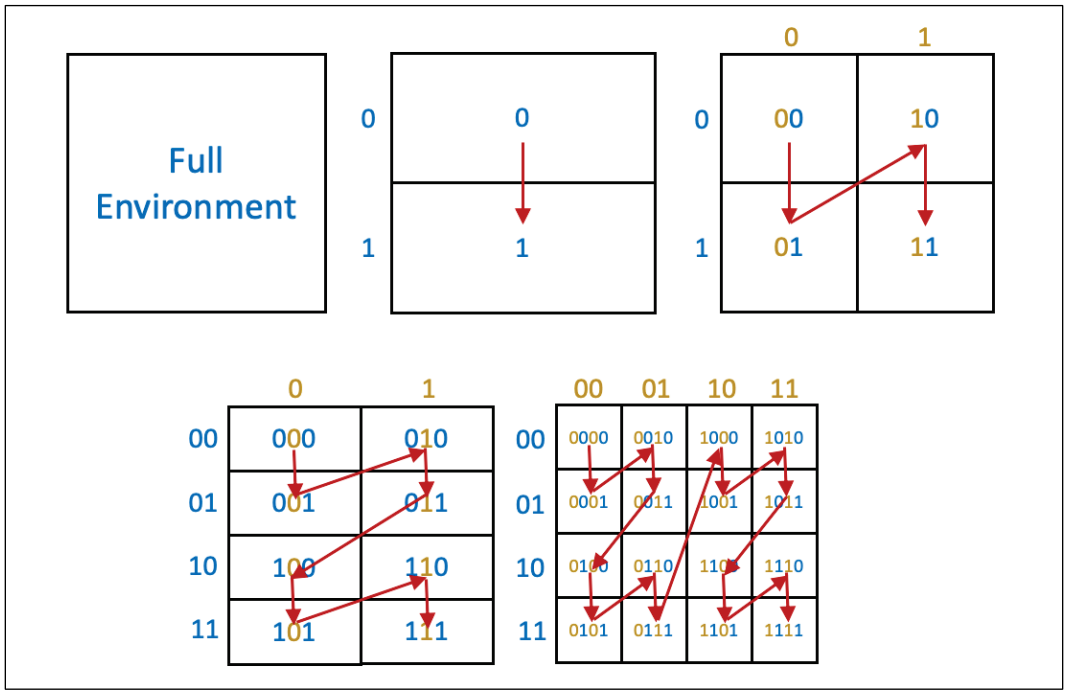
Term	Definition
Vertex	Three-dimensional geometric point used to make up a facet
Facet	Three-dimensional triangle specified by three vertices
Scene Chunk	Independent grouping of facets created from a full landscape and vegetation model
Bounding Volume Hierarchy (BVH)	A data structure created for each Scene Chunk to optimize ray casting (Goldsmith and Salmon 1987)
Geometry Engine	The geometric modeling/ray casting software component of the VESPA project

2 Morton Codes

2.1 Description

Morton codes (Morton 1966), also known as z-order curves, are used to calculate a single integral scalar value that represents a multidimensional Cartesian coordinate with orthogonal axes. For the purpose of redistributing landscape and vegetation geometry, each facet is assigned a Morton code based on its centroid coordinate. Morton codes are created by interleaving the bits of each component of a coordinate vector, resulting in a one-dimensional binary integral value. For two-dimensional coordinate systems, Morton codes are created by concatenating bits from the X and Y components in an alternating fashion, starting with the most-significant bit of each component. Figure 2 demonstrates how Morton codes can be created for two-dimensional coordinate values.

Figure 2. Subdividing geometry with Morton codes.



Generating a set of two-dimensional points from a sequence of Morton codes results in a z-shaped curve (Figure 2). These codes are useful as a hashing value for coordinate components. The Morton distance between two points can be calculated by finding the difference between the Morton codes of each point. While the Morton distance isn't an accurate measure

of cardinal spatial distance, it is a close enough approximation to provide an efficient heuristic for sorting points spatially.

2.2 Normalizing Morton codes

If using Morton codes to get an even division of the geometry as shown in Figure 2 and Figure 4, the coordinates used to create the Morton codes must be normalized to the bounding box of the geometry. This process is started by using the minimum and maximum end-corners of the axis-aligned bounding box encompassing the mesh to normalize the coordinates. The coordinate vector becomes a value between (0, 0, 0) and (1.0, 1.0, 1.0). Once the coordinate is normalized, the axis components must be converted to integers to facilitate the binary manipulation required to create the Morton code. To convert the components to an integer with enough precision to create an accurate Morton code, each double value must be multiplied by some scale factor. For this project, 2^{42} was chosen. This gives each coordinate component 42 bits of precision, which is enough precision to map the state of Texas at micrometer resolution. Computing these integer components is accomplished with the following equations:

$$X_n = \frac{X - x_{min}}{x_{max} - x_{min}} \tag{2.1}$$

$$X_{int} = \lfloor 2^{42} X_n \rfloor \tag{2.2}$$

where

X_n = Normalized X component
$x_{min/max}$ = Bounding box corners
X_{int} = Integer conversion of X_n
X = Initial world coordinate component

Equations 2.1 and 2.2 are applied to each coordinate component, not just the x-axis. The Morton code can be created once each component has been normalized and converted to a large integer value. Deconstructing the Morton code into the coordinate components can then be accomplished as long as the scale factor and the original axis-aligned bounding box are known.

2.3 Partitioning with Morton codes

Morton codes are used as an efficient means of subdividing the landscape (with vegetation) geometries used by the Geometry Engine. By selecting the first n most-significant bits of a facet's centroid Morton code—called the Morton "header"—the facets can be divided into spatially localized groups that contain facets with Morton headers of the n most-significant bits. Using longer Morton headers (higher values of n) divides the geometry into more groups with smaller enclosing volumes (Figure 2).

As mentioned previously, the algorithm used by the Geometry Engine divides the geometry into multiple HPC nodes without overflowing the memory of any single HPC node in the parallel environment. This requires a two-phase process. First, the facets from the geometry are loaded directly from a file into each MPI process in parallel. These facets may have no particular order, which means they can be non-spatially distributed. Once each facet and its associated vertices exist in the memory of an individual MPI process, the centroid of each facet is calculated and converted into a Morton code. Second, using the Morton header codes as the key, a parallel sort will be performed to group the facets according to the most-significant bits of the Morton codes. Each group can then be assigned, according to the Morton header code, to a specific MPI process. This effectively spatially sorts the geometry into its Morton header groups, which are contained on different processes. This sorting phase, Phase two, is described further in Section 2.5. Figure 3 lays out this process and Figure 4 shows how this subdivides the landscape geometry.

Figure 3. Phases of the partitioning process.

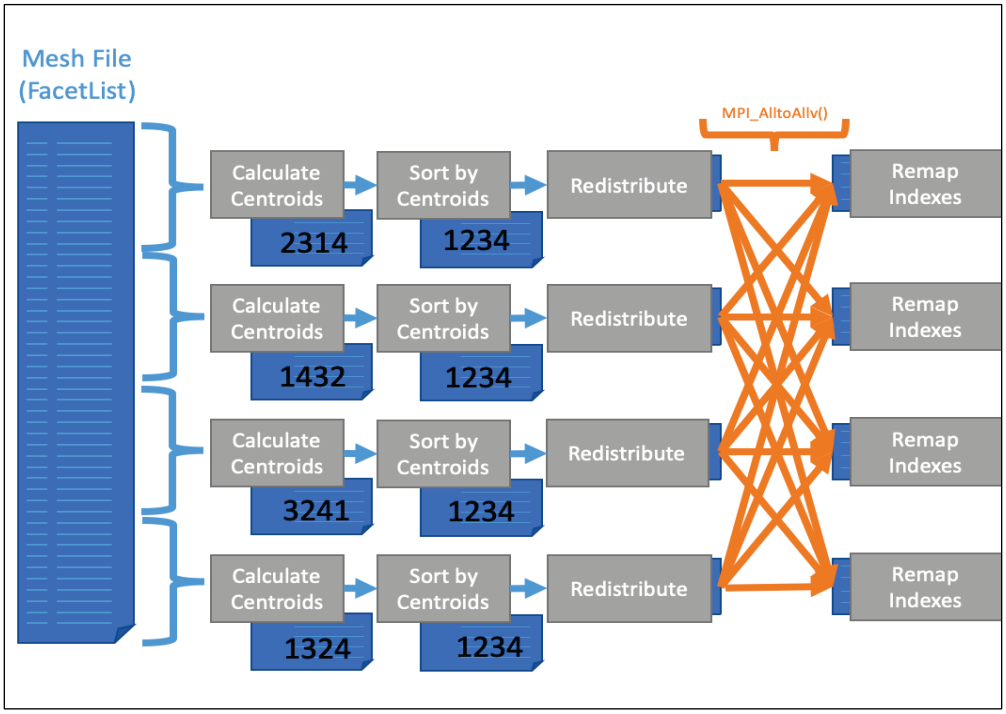
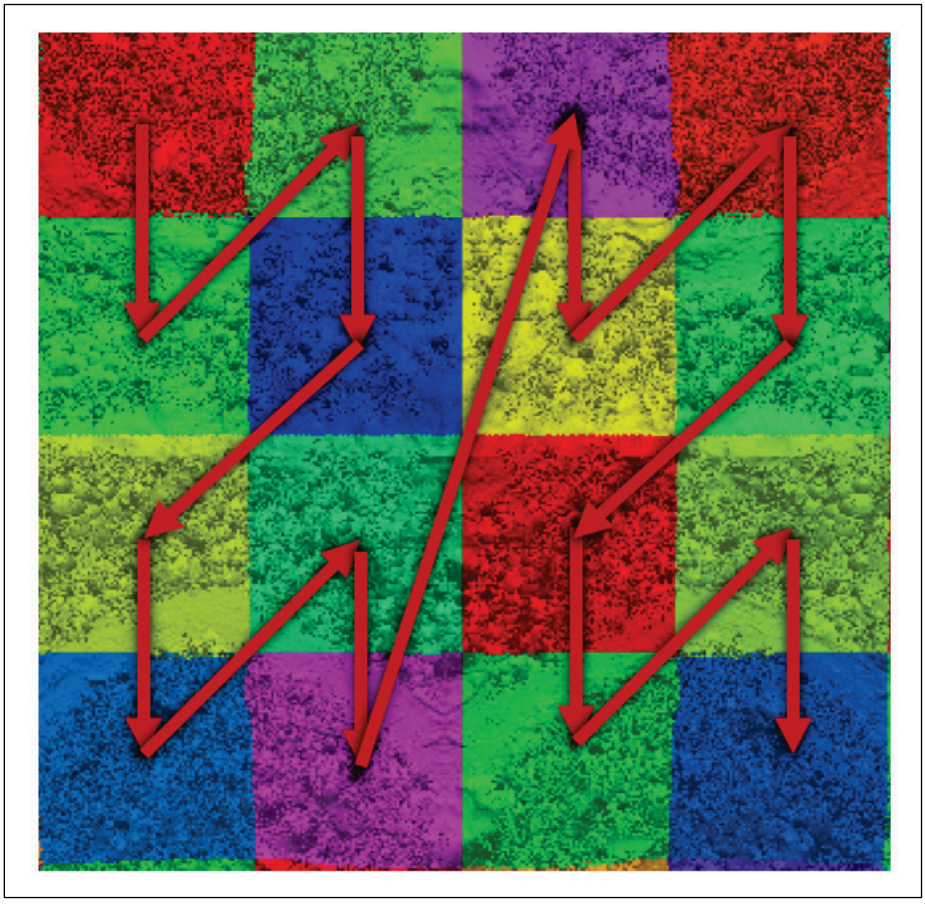


Figure 4. Landscape partitioned with Morton codes.



Note that the geometry is only divided along the X and Y axes in Figure 4. Because large landscape geometries vary far more over the X and Y directions than the Z direction, the points in the landscape are treated as two-dimensional points when converted to Morton codes, ignoring the Z component.

2.4 Fetching data from disk

The facet and vertex data have to be loaded from physical storage into the local memory of each MPI process. The MPI processes of the engine divide the facets and fetch them from the Mesh Binary file first. Each process selects a slice of the facets to load based solely on the rank of the process and calculates the offset to pull the facets from.

$$\text{slice start} = (\text{header size}) + (\text{vertex size})(\text{vertex count}) + (\text{facet size})(\text{facets per process})(\text{process rank}) \tag{2.3}$$

$$\text{slice end} = (\text{slice start}) + (\text{facet size})(\text{facets per process}) \tag{2.4}$$

The last process gets the remainder of the facets by ending its slice at the end of the file instead of at the "slice end" calculated in Equation 2.4.

2.4.1 Loading coordinate vertices

Once the facets are loaded into all of the processes, each process fetches a set of vertices based on the global indexes included in these loaded facets. This set of vertex indexes is sorted and the duplicates are removed. The vertices are then read individually from the geometry file using the global index (vertex ID) to locate the position of the vertex in the file (Equation 2.5).

$$\text{vertex offset} = (\text{header size}) + (\text{vertex size})(\text{vertex ID}) \tag{2.5}$$

Following the procedure above, the MPI processes can fetch all of the data for the vertices associated with the facets it previously gathered.

After the vertices associated with a process's facets are loaded, the indices of the vertices in the facets have to be remapped to match the position of the vertices in the list local to the process. Before this is accomplished, the list of vertices is sorted and the duplicates are removed. Then, for each facet on the process list and each of the three vertices attached to the facet,

a binary search is performed on the list of vertices to find the vertex matching the ID specified by the facet. Once the vertex is located, a new index is assigned to the facet matching the vertex's new position in the local vertex list.

2.5 Communication

Once the facets of the mesh are sorted by the Morton header codes on each local MPI process, the task becomes redistributing the facets so MPI processes can place facets in groups with an identical Morton header code, dividing the geometry into spatial chunks across the processes. This involves assigning Morton headers to individual processes and using MPI's communication interface to transfer facets from each MPI process to other processes based on the Morton header of the facet. Following this, the facets are sorted on additional Morton bits into "Scene Chunks" (Section 3.2).

2.5.1 Assigning Morton headers to processes

To ensure each MPI process ends with a set of spatial chunks (Scene Chunks, Section 3.2), the processes need a Morton header assigned, or a list of Morton headers. This is accomplished by a combination of static parameters and dynamic load balancing. First, the length of the Morton header to be used (n) must be decided. Selecting this value will provide 2^n unique values for Morton header codes of the facets. A list of Scene Chunks can then be created, one chunk for each Morton header. Note that the number of Scene Chunks may exceed the number of MPI "worker" processes. The number of bits (w) assigned to MPI "worker" processes is still limited to 2^w and the number of scene chunks per worker process is 2^{n-w}.

The value for n is selected based on a static configuration parameter, namely, a minimum number of facets to include in each Morton header. This minimum value is set in a configuration file and loaded on the partitioning program's initiation. The number of bits needed for the Morton header is then calculated as follows:

$$n = \left\lceil \log_2 \frac{\mathrm{N}}{\mathrm{m}} \right\rceil \tag{2.6}$$

where:

N = Total number of facets in the geometry
m = Minimum number of facets per chunk

After the value for n is calculated, it follows that the number of chunks will be equal to the number of unique Morton headers of length n, which is 2^n .

Once the algorithm knows how many unique Morton headers are being used, it builds an array that maps each chunk with value from $0 \rightarrow (2^n - 1)$ to an MPI process rank (ranging in value from $0 \rightarrow (2^w - 1)$). Scene chunks that begin with the same first w bits will then be assigned to an appropriate MPI process using this array, which is used later to communicate facets loaded in phase 1 (as described in Section 2.3) to the process assigned to the facet's Morton header.

2.5.2 MPI collective communication

Following the sorting of the facets by Morton header on each MPI process and the generation of the chunk distribution array, the algorithm is ready to regroup the facets among the MPI processes by chunks (see the "Redistribute" step in Figure 3). The MPI collective communication routine MPI_AlltoAllv (MPI 2015) provides a high level of control for redistributing data across all processes. This routine allows specified slices of a buffer of facets based on Morton header codes to be sent to each MPI process. The pseudo code for partitioning the facet buffers and performing the MPI_AlltoAllv call is included in the Appendix. Once this phase is completed, each MPI process contains only facets belonging to the Morton headers (chunks) assigned to that process.

2.5.3 Limitations

Although MPI's collective communication routines provide a robust, efficient approach to redistributing the facets, there are some inherent limitations. The primary limitation comes when working with the large, memory-intensive landscape models required by the VESPA project. When a large mesh is used in conjunction with relatively few MPI processes, the ratio of facets to processes becomes substantially large and the MPI_AlltoAllv call cannot deal with the extreme data sizes. Specifically, the MPI routine only deals with 32-bit integer datatypes, so when a process needs to send n facets to another process where n exceeds

the limit of a 32-bit integer (~4 million), the values of the MPI routine overflow and the program fails. There is currently not a version of MPI_AlltoAllv that supports 64-bit (long int) datatypes (Hammond et al. 2014). A workaround to this limitation at present is increasing the number of MPI processes and ensuring that the 32-bit limit is never reached.

3 The Distributed Geometry

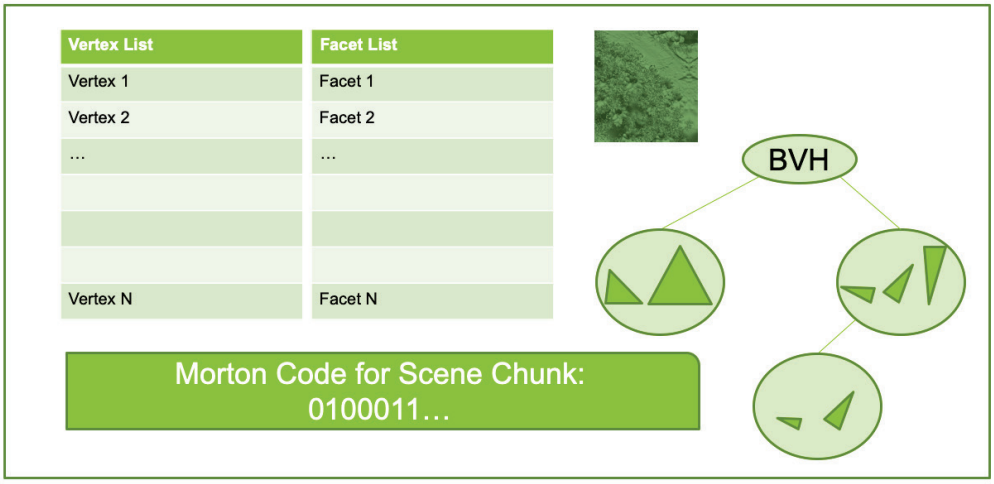
3.1 Remapping process

After the geometry is distributed across all of the MPI processes used by the partitioning program, each chunk is reorganized into a self-contained mesh, defining a mesh as an independent collection of facets and vertices. This means that the IDs (or indexes) of the facets and vertices are remapped from the original values to new local facet and vertex indexes. Since the facet structure used by the program contains ordered references to vertex IDs, each facet has to be rewritten to point to the new local IDs of the chunk's vertices. This is an identical repetition of the procedure described in Section 2.4.1. Once the facets and vertices have been rewritten with the new local IDs, the chunk can be analyzed or written out with its own mesh structure, completely independent of the original geometry it was pulled from. These independent geometry chunks are modularized into structures called "Scene Chunks."

3.2 Scene Chunks

Once each facet of a mesh has been assigned a one-dimensional hash value using Morton codes, they are split into smaller, landscape-block representations, called "Scene Chunks." A modular representation of these Scene Chunks is used to facilitate this partitioning. Each scene chunk stores data in much the same way as the algorithm might store the data if it could allocate enough memory for the whole scene model. It is comprised of a minimum axis-aligned bounding box (AABB) that surrounds the volume of the data, an array of local triangular facet structures (maintaining global indices), an array of local vertices, and a local bounding volume hierarchy (BVH), determined by a procedure that is outside of the scope of this report. The Scene Chunk representation provides I/O methods for writing and reading the geometry. Each MPI process can perform these methods on the individual Scene Chunks that belong to it completely independent of the global mesh. Figure 5 lays out an overview of these Scene Chunk objects.

Figure 5. Scene chunk modular description.



4 Results

The distributed architecture described in this report successfully partitions and performs ray casting on landscape and vegetation geometries over a Terabyte in size. The timings acquired from the partitioning step itself validate the method's ability to handle geometries larger than the memory of a single HPC compute node. These timings also demonstrate the embarrassingly parallel scalability of the method as the number of MPI processes is increased.

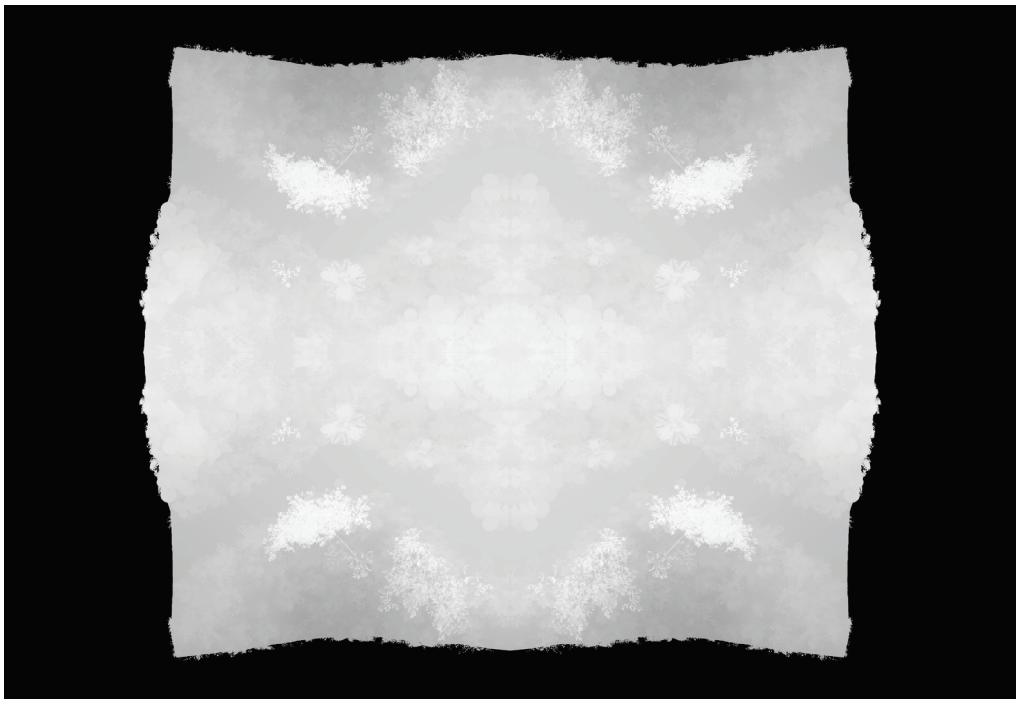
All of the tests outlined below are run on Onyx, the Department of Defense (DoD) supercomputer local to the Information Technology Laboratory (ITL). A regular compute node on Onyx has 44 cores and 128 GB of memory*.

4.1 Partitioning timings

The landscape geometry used to test this architecture was initially 7 GB in size. In order to test the partitioning methodology with an unstructured mesh larger than the memory of a single compute node (128 GB), the original 7 GB geometry was mirrored to remove gaps at the seams, and then tiled to create large landscapes. Figure 6 displays an overhead view of one tile created by mirroring the initial landscape mesh.

* ERDC DoD Supercomputing Resource Center. (n.d.). ERDC DSRC - Hardware. Retrieved November 20, 2019, from https://www.erdc.hpc.mil/hardware/index.html#onyx

Figure 6. Overhead view of landscape tile.



The largest mesh (1.1 TB in size) was created with an 8 x 8 tiling of this mirrored landscape. Intermediary meshes were also created to fully test the capabilities of the partitioning architecture.

Figure 7 and Figure 8 show the time required for the complete partitioning process with I/O and without I/O, respectively. The curves for the larger meshes are limited to running with the higher quantity of processes due to the MPI limitation discussed in Section 2.5.3.

Figure 7. Total runtime for partitioning on Onyx.

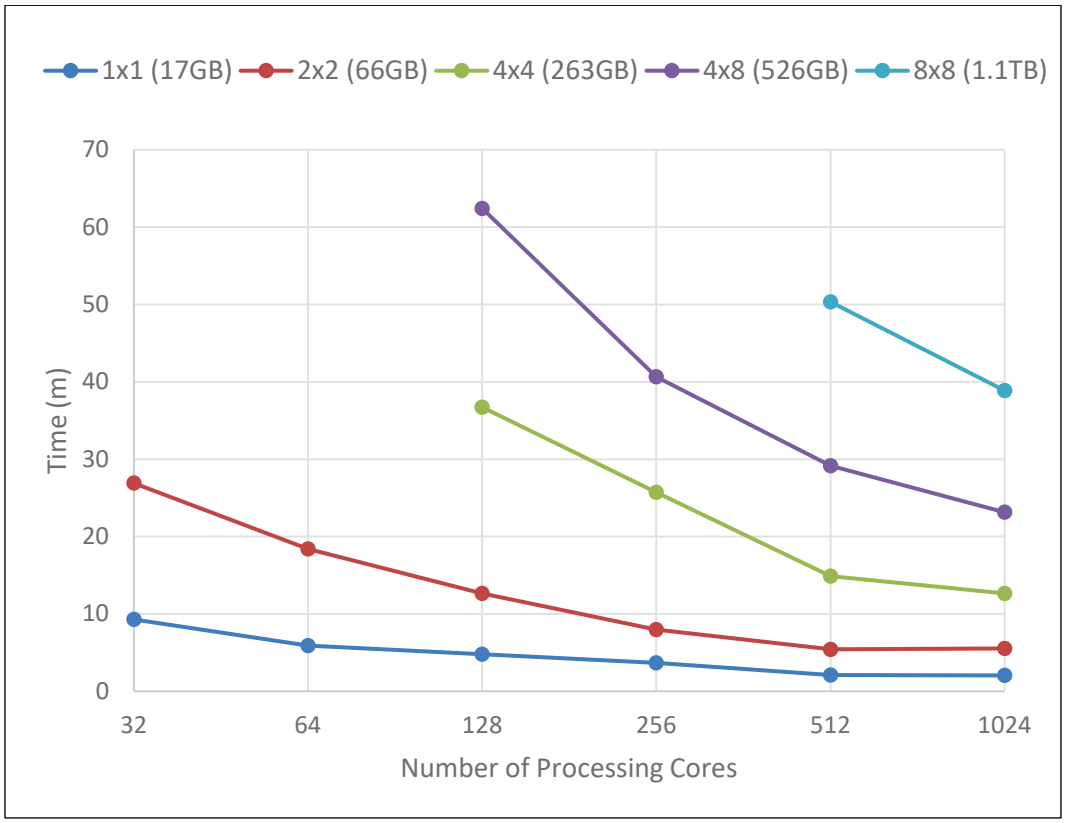
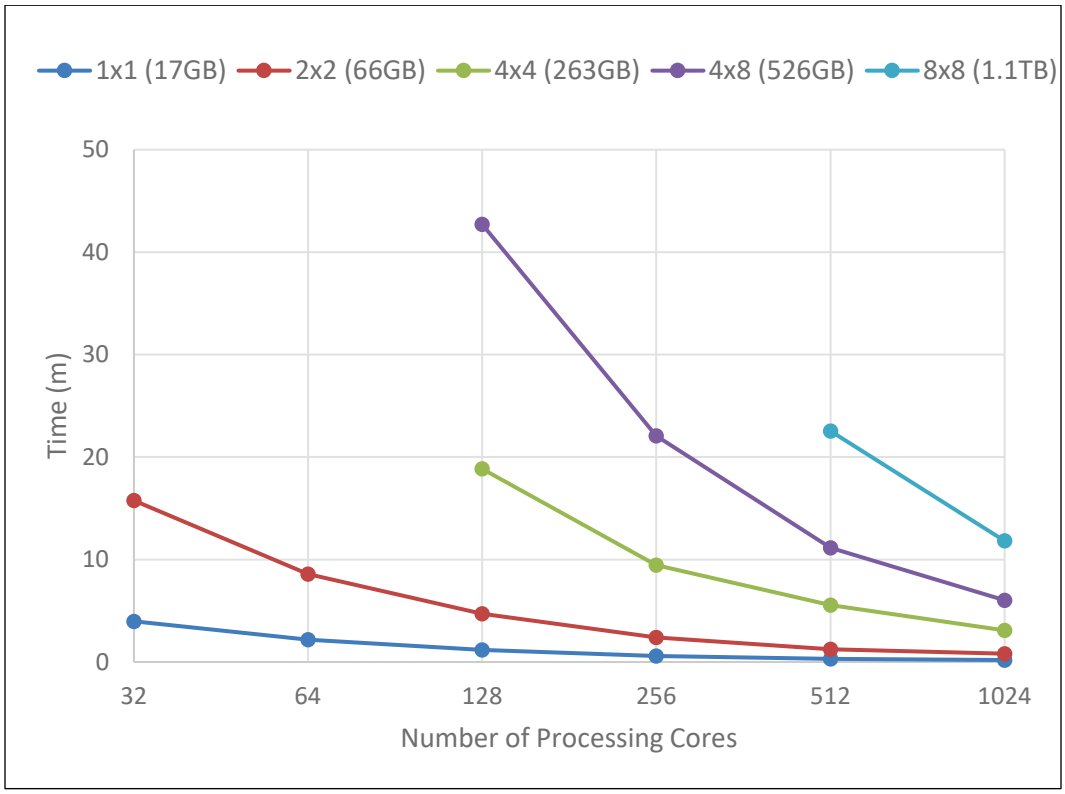


Figure 8. Processing time for partitioning on Onyx.



Both plots demonstrate the distribution methodology is able to partition landscapes that far exceed the size of a single compute node's local memory. The curves also show significant improvement in speed as the processes are increased, which verifies that the architecture is correctly distributing the load across the available processes.

To get a better understanding of the scalability of the algorithm, the relative efficiency can be determined. Efficiency is normally calculated by comparing the parallel run time to the serial run time (Kumar et al. 1994). However, this particular partitioning problem is inherently a parallel problem and cannot be run on large meshes in serial. Thus, a metric was derived for the efficiency over n runs by comparing a run i with time T_i to a run $i - 1$ with time T_{i-1}. The efficiency is the relationship between the decrease in time T_{i-1}/T_i to the increase in the number of processes, P_i/P_{i-1}. Thus, the efficiency is calculated by:

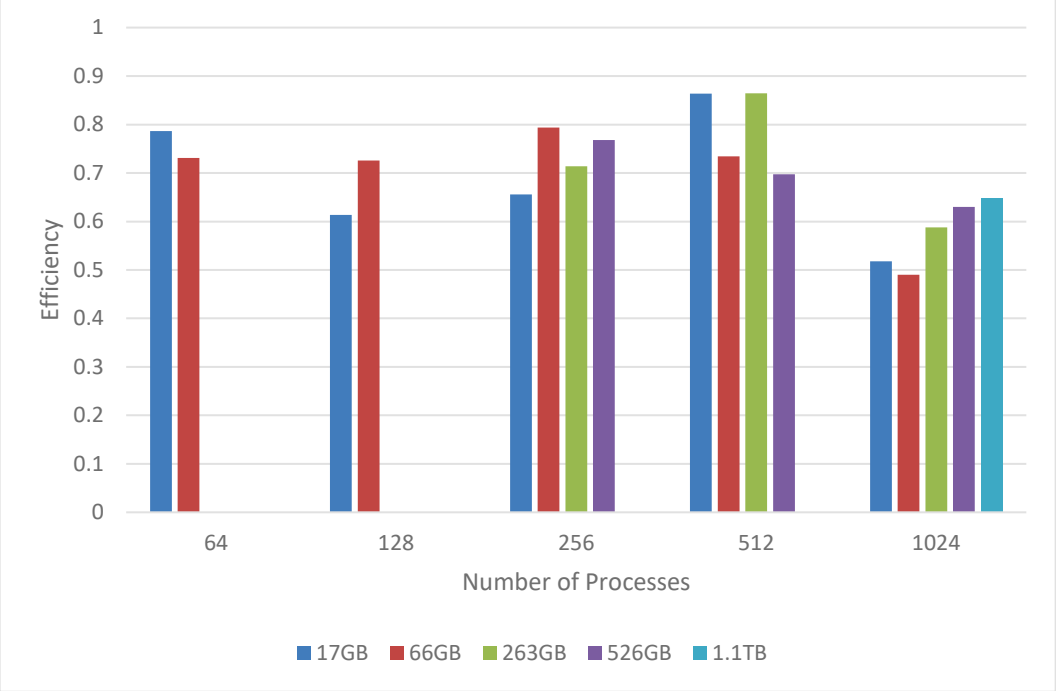
$$\frac{T_{t-1}}{T_i} / \frac{P_i}{P_{i-1}} \text{ or } \frac{T_{i-1}P_{i-1}}{T_iP_i} \tag{4.1}$$

where

P_i = Number of processes
T_i = Time corresponding to run

Figure 9 shows the values for efficiency for each size geometry as the number of MPI processes is increased.

Figure 9. Efficiency of partitioning procedure.



This plot further demonstrates the scalability of the architecture. Doubling the number of MPI processes provides between 60%-70% speedup for most of the runs. This only starts to diminish once it reaches 1,024 processes.

5 Conclusion

The Morton code partitioning method described in this report provided the VESPA project with the capability to process landscape geometries that exceed the memory of a single HPC compute node. Morton codes provide an efficient heuristic for spatially sorting the geometric facets, allowing MPI collective communication to partition the geometry into spatial chunks across parallel memory. This method showed scalable timings for partitioning geometric data. Overall, the partitioning process has been shown to increase ray casting times in a similarly scalable fashion.

This geometry distribution capability has many applications to the VESPA project. Parallel ray casting can be used to perform radiative transfer modeling to populate the landscape model with temperature and energy flux data. Ray casting was also used by the imaging sensor model of the VESPA workflow to fetch the energy data needed to generate infrared imagery. In addition to ray casting, this method could be adapted to perform other forms of parallel analysis involving large-scale landscape geometries.

5.1 Limitations

During the development of the distributed algorithm for processing landscape geometry, limitations in the method used were discovered. As mentioned in Section 2.5.3, one such limitation is the amount of data that can be sent between processes using MPI_AlltoAllv. This routine uses 32-bit integer datatypes to define the size of the buffers used to communicate, so if the size of the buffer exceeds the limit of a 32-bit integer, an overflow will occur causing the software to crash. Currently, an overflow is prevented by increasing the number of MPI processes, reducing the amount of data that has to be sent/received from individual processes.

5.2 Future work

While there are other components of the Geometry Engine and the VESPA project as a whole that will continue to be developed to utilize the distributed architecture described in this paper, there are some specific features that will be implemented and developed on the partitioning architecture itself.

5.2.1 Eliminating limitations

A particular area of improvement would be eliminating the MPI_AlltoAllv limitation described in Section 2.5.3. This could be done by finding a replacement for MPI_AlltoAllv that allows for buffer lengths exceeding the 32-bit integer range. However, an alternative that could be explored would be defining an MPI datatype representing a chunk of facets. In fact, this is how MPI recommends working around the 32-bit count limitation (Hammond et al. 2014). Because the buffer lengths are specified in terms of the number of MPI structures in the buffer, defining a larger MPI data structure would allow more data to be transferred with smaller count values. This would keep the counts in the 32-bit range while transferring more than 2^{32} facets.

5.2.2 Load balancing

Different load balancing options will be explored for distributing the landscape geometry more evenly across HPC compute nodes. One such option is using a more dynamic method for determining the length of the Morton headers for Scene Chunks. Landscape chunks with heavy vegetation could be subdivided using longer Morton headers than chunks with light vegetation, distributing the geometry data more evenly.

5.2.3 Optimizing

Parametric studies will also be conducted to determine the optimum value for m in Section 2.3, the minimum number of facets per Scene Chunk. Running an optimization scheme across this parameter would be useful in establishing the Scene Chunk size resulting in the fastest ray casting. There are additional parameters, particularly in the BVH construction process, that could be optimized to increase the ray casting performance.

5.2.4 Data compression

There is potential to incorporate data compression techniques into the Geometry Engine to process large landscape geometries when the number of HPC compute nodes available is small. Vertices can be compressed using Morton codes, reducing a three-dimensional coordinate vector from 24 bytes to 16 bytes, assuming 42-bits are used for each Morton component as specified in Section 2.2. This would reduce the size of the vertices by 33%. Similar compressions could be explored for the facet and BVH data structures as well.

References

Goldsmith, J., and J. Salmon. 1987. "Automatic creation of object hierarchies for ray tracing." *IEEE Computer Graphics and Applications*, 7(5): 14–20. https://doi.org/10.1109/MCG.1987.276983.

Hammond, J. R., A. Schäfer, and R. Latham. 2014. "To INT_MAX... And Beyond!: Exploring Large-count Support in MPI." Proceedings of the 2014 Workshop on Exascale MPI, 1–8. https://doi.org/10.1109/ExaMPI.2014.5.

Howington, S. E., J. R. Ballard Jr, N. V. Kala, A. C. Trautz, M. D. Bray, M. W. Farthing, and A. M. Hines. 2019. "Exploitable synthetic sensor imagery from high-fidelity, physics-based target and background modeling." In *Target and Background Signatures V*, *11158*, 1115809.

Kumar, V., A. Grama, A. Gupta, and G. Karypis. 1994. Introduction to parallel computing: algorithm design and analysis. Benjamin Cummings/Addison Wesley, Redwod City.

Morton, G. M. 1966. A computer oriented geodetic data base and a new technique in file sequencing. https://domino.research.ibm.com/library/cyberdig.nsf/papers/0DABF9473B9C86D48525779800566A39/$File/Morton1966.pdf

Watt, A., and M. Watt. 1992. "Polygonal representation of three-dimensional objects." In P. (Brown U. Wegner (Ed.), *Advanced Animation and Rendering Techniques* (pp. 4–5). New York, NY, USA: Addison-Wesley Publishing Company.

Appendix A: Pseudo Code for Distribution Routine

```
Declare variables send_lengths, receive_lengths,
send_displacements, receive_displacements

Chunk distribution array is defined as chunk_dist

For every facet on this process

    Get the rank set in chunk_dist at the index of the facet's
Morton header

    Increment send_lengths value at the index defined by that
rank

End For

Populate receive_lengths by redistributing the send_lengths array
among all processes using MPI_AlltoAll

Initialize send_displacements and receive_displacements first
value to 0

For i in the list of MPI ranks (skipping the first rank)

    Set the displacement in send_displacements for this rank to
    be the displacement of the previous rank plus the length of
    the previous rank (from send_lengths)

    Repeat for receive_displacements and receive_lengths

End For

Redistribute facets with MPI_AlltoAllv using send_lengths,
send_displacements, receive_lengths, and receive_displacements as
the parameters
```

Acronyms

AABB	axis-aligned bounding box
BVH	Bounding Volume Hierarchy
DoD	Department of Defense
HPC	High Performance Computer
I/O	Input/Output
ITL	Information Technology Laboratory
MPI	Message Passing Interface
VESPA	Virtual Environment for Sensor Performance Assessment

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
April 2020	Final	

4. TITLE AND SUBTITLE

Using Morton Codes to Partition Faceted Geometry: An Architecture for Terabyte-Scale Geometry Models

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Robert H. Hunter, Barry C. White, Reena R. Patel, and Jerrell R. Ballard, Jr.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC/ITL SR-20-3

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

MIPR Number HQ0642033036

14. ABSTRACT

The Virtual Environment for Sensor Performance Assessment (VESPA) project requires enormous, high-fidelity landscape models to generate synthetic sensor imagery with little to no artificial artifacts. These high-fidelity landscapes require a memory footprint substantially larger than a single High Performance Computer's (HPC) compute node's local memory. Processing geometries this size requires distributing the geometry over multiple compute nodes instead of including a full copy in each compute node, the common approach in parallel modeling applications. To process these geometric models in parallel memory on a high-performance computing system, the Geometry Engine component of the VESPA project includes an architecture for partitioning the geometry spatially using Morton codes and MPI (Message Passing Interface) collective communication routines. The methods used for this partitioning process will be addressed in this report. Incorporating this distributed architecture into the Geometry Engine provides the capability to distribute and perform parallel ray casting on landscape geometries over a Terabyte in size. Test case timings demonstrate scalable speedups as the number of processes are increased on an HPC machine.

15. SUBJECT TERMS

Geometrical models
High performance computing
Parallel processing (Electronic computers)
Electronic data processing
Computer architecture

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	SAR	31	