Digital Lunar Exploration Sites Unreal Simulation Tool (DUST)

Lee Bingham
NASA,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Lee.K.Bingham@nasa.gov

Jack Kincaid
METECS,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Jack.A.Kincaid@nasa.gov

Benjamin Weno
METECS,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Ben.Weno@nasa.gov

Nicholas Davis
METECS,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Nicholas.R.Davis@nasa.gov

Eddie Paddock
NASA,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Eddie.Paddock@nasa.gov

Cory Foreman
METECS,
NASA Johnson Space Center
2101 NASA Parkway
Houston, TX 77058
Cory.D.Foreman@nasa.gov

Abstract— **NASA's future Artemis missions to the Moon seek to explore areas around the Lunar South Pole. Though humans have previously set foot on the lunar surface, the proposed region provides unique and challenging environments that require insight and investigation prior to arrival. Several teams throughout the agency are performing this site and mission planning, design, and analysis to support areas like the Human Landing System (HLS), surface mobility, habitation elements, and scientific exploration.**

The NASA Exploration Systems Simulation (NExSyS) team at Johnson Space Center is developing a graphical environment of the Lunar South Pole region. Lunar terrain information collected from the Lunar Reconnaissance Orbiter (LRO) is compiled and made available through Johnson Space Center's Digital Lunar Exploration Sites (DLES) data sets. The DLES data is used to build this graphic environment. The process of ingesting and accurately modeling this information in a meaningful way for analysis creates its own challenges such as generating a performant model from the source data and the application of curvature. Additionally, the area around the Lunar South Pole experiences different lighting conditions than those observed from the Apollo missions. The need to use the lunar environmental data products provided by DLES combined with the capability to calculate date specific ephemerides in real-time has given rise to the development of the DLES Unreal Simulation Tool (DUST). DUST incorporates augmented terrain from the DLES product into a desktop application that allows exploration of the Lunar South Pole region and its complex lighting conditions. DUST leverages advanced capabilities in the recently released Unreal Engine 5 renderer by Epic Games such as double precision for positioning of planetary bodies and surface elements, multiple infinite light sources to represent the Sun and eventually Earthshine, high resolution shadow maps for dynamic shadow accuracy, real-time software ray-tracing for multi-surface bounce lighting to render sunlight reflected off surface elements and terrain features, and performance optimized level of detail shifting as the eyepoint changes in a scene.

This paper details the DUST application, the technologies of the engine platform that enable scientific and engineering analysis, the unique techniques and processes developed to consume the DLES data sets, and how the tool is being used to support the Artemis program.

TABLE OF CONTENTS

INTRODUCTION

NASA's Artemis program, in collaboration with its commercial and international partners, will establish the first long-term human-robotic presence on and around the Moon. This program will demonstrate new technologies and capabilities necessary to assist future exploration, provide the opportunity to study the Moon, and inspire a new generation. As part of the models and simulations (M&S) in support of the Artemis program, the Digital Lunar Exploration Sites (DLES) products are intended to provide essential lunar environmental data [1].

The ability to accurately model the lunar environment is crucial in developing simulations for the various elements and aspects of the Artemis missions. There are numerous applications and tools that are used around the agency to visualize the Moon. Moon Trek, developed by NASA's Jet Propulsion Laboratory, provides a web-based tool that presents 3D visualization of imagery data from hundreds of lunar data products [2]. Lunaserv was developed by Arizona State University as part of the Lunar Reconnaissance Orbiter Camera (LROC) project to serve as a map server to render non-Earth datasets [3]. These tools have proven to be valuable for analysis and site selection, but do not focus on surface-level features and how lighting plays a critical role on operations and exploration.

The areas of interest around the Lunar South Pole region experience challenging lighting conditions vastly different than those seen during the Apollo missions. The NASA Exploration Systems Simulation (NExSyS) team tasked the development of the DLES Unreal Simulation Tool (DUST) that can leverage the pedigree of modern commercial rendering engines to provide a real-time high-fidelity visualization of the lunar environment data with representative lighting conditions that could be used to explore surface-level features and conditions. This paper provides an overview and design goals of the DUST application and its capabilities; the methods of generating the terrain; how the lunar environment is modeled; and finally, future work and how the DUST application is being used in support of the Artemis program.

DESIGN GOALS

NASA has a diverse development history of visualization software for training and analysis. From virtual reality training [4], to virtual environment data products [1], and scientific visualization [5], NASA has developed a wide array of rendering applications to aid in our pursuit of space exploration. The DLES Unreal Simulation Tool (DUST) is a novel advancement in visualization technology that provides unprecedented scale and level of detail through a combination of new technologies and previous lessons learned.

DUST was initially conceived to provide a tool that could quickly visualize the DLES data products in concert with date specific lighting. It quickly grew to encompass several additional capabilities to support site and mission planning for lunar architecture and Artemis Base Camp (ABC) [6]. To reach a broader community of engineers and scientist, the DUST application was designed to provide a high-fidelity visualization and analysis tool while remaining performant on commodity workstations and compatibility for Windows, Mac, and Linux users. Ultimately as this product matures it will provide a framework with the ability to share and collaborate throughout the agency as well as commercial partners.

CAPABILITIES OVERVIEW

The DUST application integrates a variety of tools and features designed to provide analysis capabilities of the lunar terrain and support in preparation of the Artemis missions. These tools provide interactions within the simulation such as lunar rock placement and manipulation, celestial body positioning, and visualization of date specific lighting at sites of interest. Additional tools have been developed to support site planning and analysis. These include features such as topography and slope visualization, the ability to import traverses and edit in real-time, and the implementation of a line-of-sight communication model.

SPICE Integration

SPICE is a toolkit developed by NASA to track positions and orientations of planetary bodies and spacecraft. The toolkit provides an API that reads kernel datasets containing information about the tracked objects over a timeframe. After the datasets are loaded, a datetime can be passed to the API to get the positions of the given object. In order to integrate this API within Unreal Engine, we use the MaxQ Spaceflight Toolkit [7]. In DUST, we use the toolkit to track the positions of the Sun and Earth in relation to the lunar south pole as shown in Figure 1. We can also visualize spacecraft trajectories via SPICE ephemerides such as the proposed near-rectilinear halo orbit (NRHO) of the Gateway stack. The relevant kernels are packaged with the build and loaded with the application. This enables accurate positions and lighting angles for the proposed landing and mission timeframes. A user interface (UI) can be used to select any point in the data, and playback at multiple rates is supported.

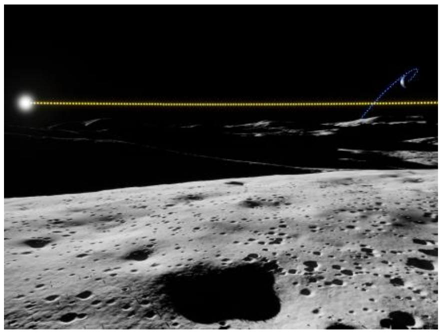


Figure 1. Sun/Earth Trails

TRICK Integration

TRICK is a NASA developed open-source simulation environment that provides an architecture for simulation development [8] [9]. TRICK has supported the development of high-fidelity engineering simulations at NASA in a variety of applications over the past several decades [9]. In DUST the user can connect to a TRICK simulation over a socket, enabling rover control on the lunar surface. The Sun and Earth position can also be synchronized to the TRICK simulation, which can be used to replicate visuals with other engines connected to the same simulation.

PODB Integration

PODB (Persistent Object Database) is a NASA generated database with a web API wrapper that stores positional data of persistent synthetically added lunar objects such as rocks and craters. Synthetic features of the lunar surface are used to enhance surface details for low resolution DEM data. The DUST PODB integration tool shown in Figure 2. **PODB Integration Tool**, queries the PODB API for rock data in any designated region, and stores the results in a file within the build of the application. This provides rock loading capability in areas of interest without having to connect to PODB when the application is deployed.

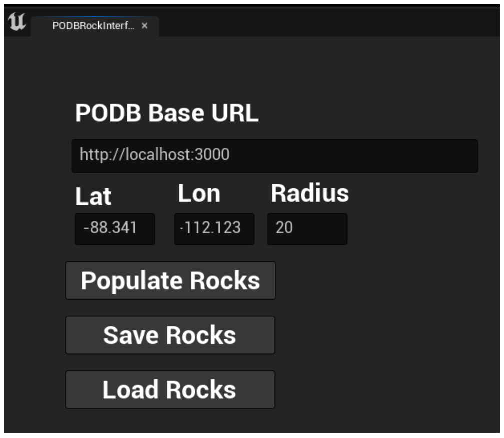


Figure 2. PODB Integration Tool

Tools

Rover Traverse Visualizer—DUST can parse rover traverse data and display it on the simulated lunar surface. This allows for verifying terrain and lighting conditions along the traverse at any point. Additionally, the user can modify and export the traverse to a .json or .geojson file format. Pins can be placed along the traverse and used to calculate moving average velocity. Dwell periods can be established at individual pins to force the rover to wait for the specified duration before continuing the traverse. Figure 3. **Rover Traverse** shows a traverse displayed on the lunar surface. There are two methods of traverse visualizing in DUST. The first utilizes the rendered spline as a path for a rover model to follow. The rover's speed can be adjusted, and the traverse rate can be scaled. This method provides a static rover model that follows the traverse path with a follow camera, lighting control, regolith particle effects, and track decals. The secondary method for traversing the visualized path utilizes the Unreal Engine Chaos physics to simulate a physical rover following the traverse. In either traverse method, information such as time stamps and velocities are calculated and displayed to the user.

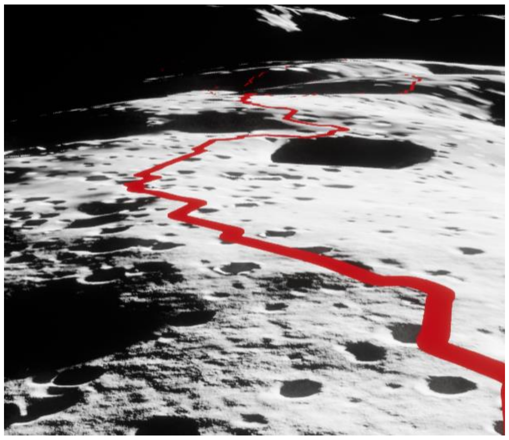


Figure 3. Rover Traverse Tool

Communication Visualization—Communication towers can be placed on the surface, and their range and occlusion can be visualized to determine where on the terrain the signal would reach with the specified tower configuration. Up to two towers may be visualized at once, as well as communications with the Gateway station and Earth. Figure 4 shows the visualization of a 10 m tall communication tower on the outside of a large crater. The blue regions indicate areas of limited signal while red indicates strong signal strength. All communication visualization is calculated via line of site and provides a visual representation with no signal metrics currently calculated.

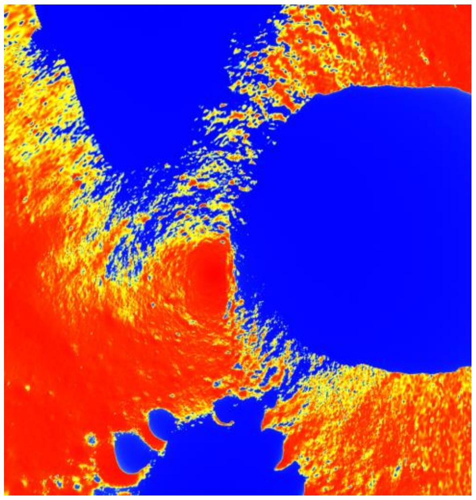


Figure 4. Communication Visualization Tool

Heatmap Visualization Tool—Heatmap images can be loaded into DUST to be displayed on top of the terrain. The scale and width of the image can be modified at import to support additional heatmaps.

Topography Analysis Tool—DUST can generate slope map analysis with contour lines overlaid across all rendered lunar terrain. Figure 5 shows how the slope map is visualized, with red areas being more extreme slopes, and green areas being flatter. The contour lines are also visible. Additionally, elevation map data can be automatically calculated and displayed in place of slope data as seen in Figure 6.

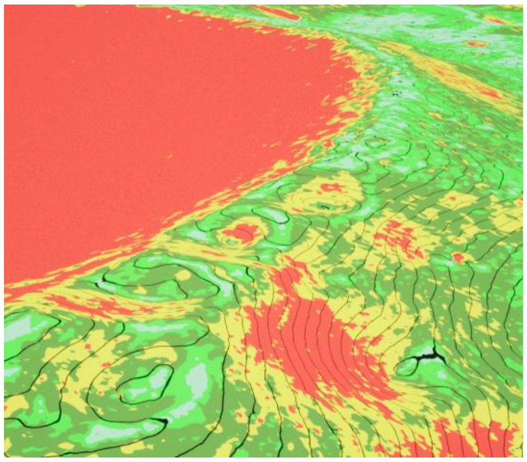


Figure 5. Slope Map Visualization

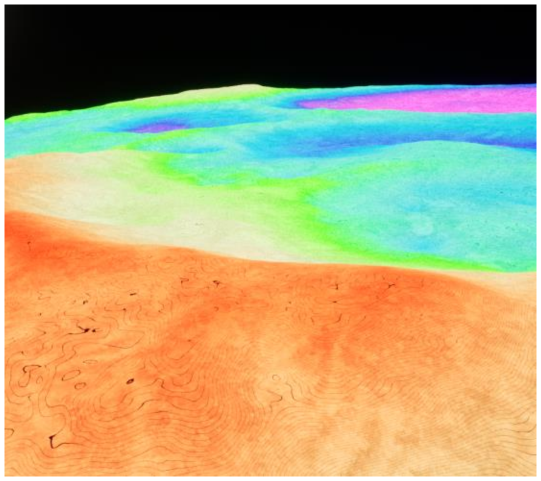


Figure 6. Elevation Visualization

3D Measuring Tool—Provides 3D measuring capability in meters between user specified locations. Points can be placed anywhere on the terrain or on objects, as seen in Figure 7 where it is utilized to measure the diameter of a crater.

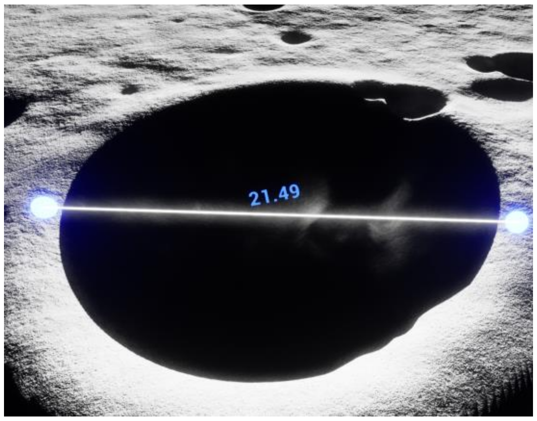


Figure 7. 3D Measuring Tool

Terrain Generation

Rendering real world data in a virtual environment is an application with uses spanning a variety of industries. Terrain generation is a complex process for which multiple techniques of accommodating varying resolution data in a virtual environment have been developed *[10] [11]*. In this section we present three methods of lunar terrain generation from LRO DEM data. Each method incorporates different features of Unreal Engine as well as various approaches to multi-resolution handling. Figure 8 shows an example of generated lunar terrain within DUST.

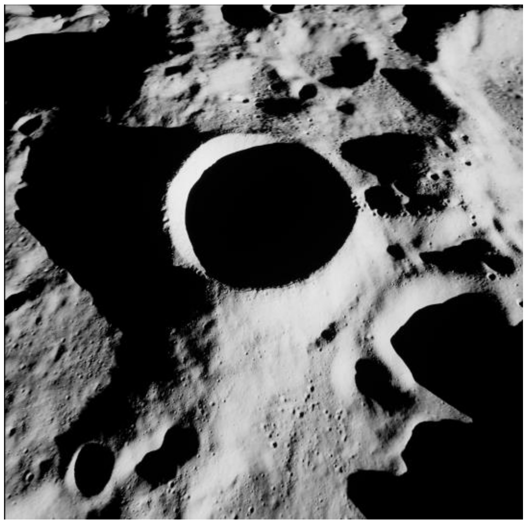


Figure 8. Generated Terrain

Utilizing Terrain Data

The terrain data used for DUST is sourced from the NASA Lunar Reconnaissance Orbiter (LRO) instruments and are stored in Digital Elevation Models (DEM). DEMs store topographic data in a texture file with an associated coordinate reference system (CRS) *[12]*. The Geospatial Data Abstraction Software Library (GDAL) is a raster manipulation tool that allows us to directly convert the DEM files to readily consumable game engine file formats like PNGs *[13]*. GDAL is also utilized within Unreal engine to read the DEMs directly via the UnrealGDAL plugin *[14]*.

Lunar Unreal Landscapes

Unreal Engine 5 has a built-in terrain generation tool referred to as the Landscape Mode. This feature allows users to easily create worlds via importing source data or using terrain modification tools to sculpt a new landscape from scratch. Using this tool, we can visualize lunar terrain from converted heightmaps. Figure 9 displays the grid-like wireframe generated by the Unreal landscape system.

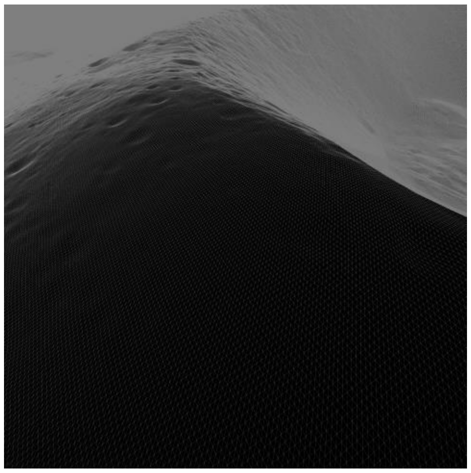


Figure 9. Landscape Wireframe View

Lunar Landscape Import—The Unreal Landscape mode supports heightmap imports in the form of PNG file format. GDAL is used to generate these PNGs from a DEM, which are then directly imported into the engine with the Landscape Mode. To ensure an accurate elevation scale a conversion must be incorporated from the DEM's Float32 scale to the Unreal Landscape Int16 scale ranging from -256 to 255.992 *[15]*.

$$(min\_elevation + max\_elevation) * 100/512 \qquad (1)$$

Equation 1 converts the height scale from the DEM's original scale to Unreal Engine's while maintaining relative positions *[15]*. It is possible to lose precision for Float32 DEMs which results in artifacts appearing on the generated landscape. This can be mitigated by dividing the DEMs into tiles resulting in a decreased range between the minimum elevations and maximum elevations.

Lunar Landscape Tradeoffs—The Unreal landscape mode is a versatile tool with numerous practical applications. It allows for stability, constant updates and support, as well as ease of use with new imports taking only a few clicks. However, there are also several downsides to the landscape system when used to generate lunar terrain. The lack of elevation precision hinders our ability to achieve engineering level accuracy in the scene and detracts from the Engine's recent improvement of general double precision support. Additionally, after the landscape models are generated, they are difficult to modify without using the in-engine landscape modeling tools which are generally effective in producing artistic or practical results with limited ability to replicate real features. This prevents us from accurately incorporating lunar curvature on the x,y, and z axes since the curvature cannot be baked into the DEM and represented in the imported PNG. Without curvature our Unreal Engine terrain does not exactly reflect our other simulations and prevents additional validation and resource sharing between the engines.

Lunar Nanite Meshes

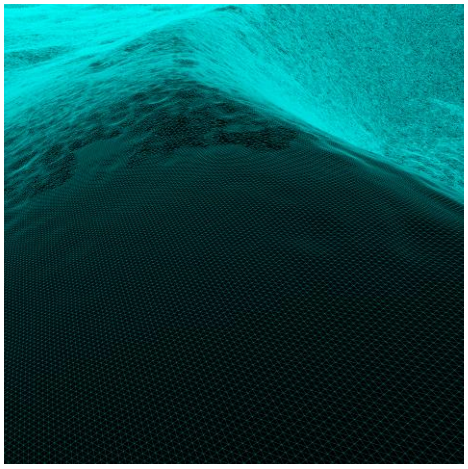
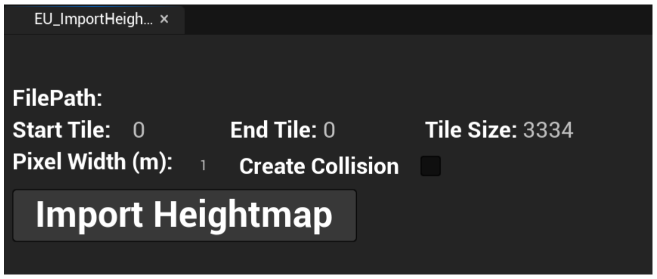


Figure 10. Nanite Wireframe View

Nanite is Unreal Engine 5's new virtualized geometry system. It provides highly compressed and performant 3D meshes through a new model format and rendering system. As the models are imported, they are broken into clusters and groups which are continuously exchanged during runtime to provide level of detail (LOD) transitions without noticeable artifacts *[16]*. We developed a method to generate vertices and triangles directly from the DEMs and use them to create an Unreal Engine static mesh with Nanite enabled. This allows us to create massive areas of lunar terrain with minimal impact on performance. Figure 10 displays what the Nanite mesh's wireframe looks like. Nanite's level of detail handling is visible further from the camera.

Converting DEMs to Nanite Mesh—To assist with DEM importing we developed a simple UI shown in Figure 11.

This UI provides fields for the file path and tiling parameters. Tiling is necessary to reduce DEMs greater than 2 GB into manageable pieces. If the tile size exceeds hardware specifications, the system will utilize an excessive amount of virtual memory causing Unreal Engine to crash unexpectedly. The recommended tile size for systems with at least 32 GB of random-access memory is 3334 by 3334.



EU_ImportHeigh... ×

FilePath:

Start Tile: 0 End Tile: 0 Tile Size: 3334

Pixel Width (m): 1 Create Collision

Import Heightmap

Figure 11. Nanite Mesh Generation UI

After the user specifies a file path and tile parameters, they need to determine the pixel width of their DEM. This can be done by using a simple GDAL command: *gdal_info -stats "filename"*. Finally, the user has the option to create a corresponding collision mesh for each tile. When the Import Heightmap button is selected, the heightmaps must first be parsed using an external plugin called UnrealGDAL which provides access to GDAL commands from within Unreal Engine *[14]*. Using UnrealGDAL the raster is read into a Float32 array and any non-data values are substituted with zero. UnrealGDAL also provides the DEM dimensions. Once the process is completed it writes out each tile as a separate static mesh, all sharing the same origin. An output of a 300 mb converted DEM to 9 Nanite static mesh tiles is shown in Figure 12.

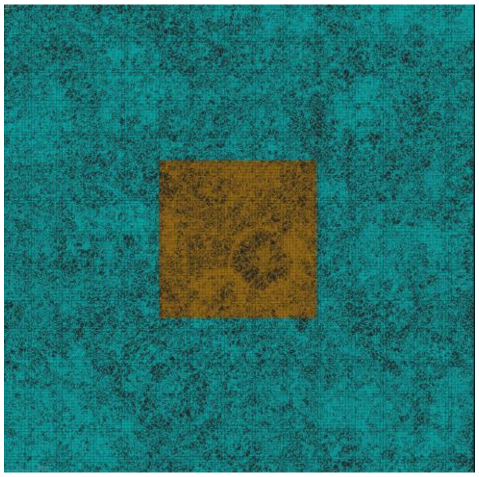


Figure 12. Tiled Nanite Mesh DEM

Mesh Generation from Heightmap Data—Using the DEM dimensions and elevation data we create the mesh information needed to define a 3D model representing the heightmaps. The dimensions allow us to create an appropriately sized grid of float vector vertices with the z-axis representing height. The stored float elevations are applied iteratively to each point's z-axis on the grid and stored in an array of vectors.

Applying Lunar Curvature—Before the elevations are applied to the array, they are first converted to our coordinate frame and lunar curvature is applied. When applying lunar curvature, we complete a planar to spherical transformation on the x, y, and z components of each vertex. This transformation assumes the center of the sphere is located at the center of our lunar reference with a radius equal to the Moon's approximate radius of 1,737,400 meters. To apply the curvature, we create a vector from the center of the Moon reference to each point on the planar grid and subtract the reference sphere's center. This vector is then normalized and extended by the lunar radius plus the heightmap's elevation value at that point. Finally, the reference sphere's center is added. These steps are represented in Equations 2,3, and 4 below.

$$\hat{V}_{norm} = \frac{\langle x-M_{centerX}, y-M_{centerY}, -M_{centerZ}\rangle}{|\langle x-M_{centerX}, y-M_{centerY}, -M_{centerZ}\rangle|} \tag{2}$$

$$\vec{V}_{extended} = \hat{V}_{norm} * (radius + elevation) \tag{3}$$

$$\vec{V}_{centered} = \vec{V}_{extended} + \vec{M}_{center} \tag{4}$$

Mesh Generation—The vertex array is used to calculate mesh triangles and normal vectors. The mesh triangles are represented as an array of integers each corresponding to an index of the vertex array. After generating the mesh data, it is forwarded to a custom function that is based on an Unreal Engine provided Static Mesh implementation. This function builds the necessary file format and rendering structure required by Unreal to be saved as their custom file format, a static mesh .uasset. Additionally, it enables Nanite initializing the required compression and formatting. Once the meshes are processed, they are saved to the Unreal Engine file system and can be used in any future level or project.

Lunar Nanite Mesh Tradeoffs— Nanite is a powerful tool that allows for rendering on an unprecedented scale for open world games or simulations involving hundreds of kilometers of terrain. With our lunar Nanite meshes we can render over 150 square kilometers of terrain at 15 m per pixel resolution with little impact on frame rate. Runtime performance is not the limiting factor when it comes to Nanite. Memory usage presents an issue when utilizing lunar Nanite meshes. As more meshes are generated, they increase the project and build size even with the efficient compression applied by Nanite. Additionally, rendering the meshes in the scene takes an increasing amount of random-access memory (RAM). Therefore, we can create high-fidelity lunar scenes with accurate terrain data and curvature, however they require intensive hardware restrictions. Currently Nanite PC support is limited to Windows OS with DirectX12 and Nvidia graphics cards *[16]*. In addition to the platform restriction of

Nanite, due to the scale of our scene, users need at least 32 GB of RAM and an NVIDIA GeForce Graphics Card 2000 series or better to launch the application. The cost of the meshes is increased depending on the amount of collisions needed. Generating collisions over the entire scene with high fidelity is not practical forcing us to limit collisions to areas where they are most needed. However, even with these limitations lunar Nanite meshes are extremely powerful with numerous potential uses. They are optimized to work with Unreal Engine's Virtual Shadow Maps, which provide performant shadows across hundreds of kilometers of terrain. Therefore, without the memory or platform constraint, it is possible to create lunar scenes with tens of millions of vertices that minimally affect the overall frame rate and rendering cost. To provide a solution to these limitations we decided to pursue a dynamically updating terrain approach.

Lunar Clipmaps

The terrain implementation currently implemented in the DUST application utilizes a simplified runtime clipmap approach. Our DEMs are loaded into memory and provide data to our rendered mesh to dynamically position the vertices.

Clipmaps—Geometry clipmaps refer to a variable resolution terrain rendering approach developed by Losasso and Hoppe in 2004 *[10]*. Losasso and Hoppe present a leveled terrain rendering grid where the resolution decreases as the distance from the viewpoint increases. During runtime, their grid would receive heightmap data or noise data and assign it to the mesh vertex buffer. The grid would follow the viewpoint and update the mesh vertex buffer as the viewpoint moved *[10]*. We adapted the clipmap rendering approach for our Unreal Engine lunar terrain.

Mesh Generation—When the application is first started, a flat clipmap grid is generated. During runtime, the grid is referenced each time a terrain update is needed. Each vertex in the grid is moved to its proper position with the DEM information and curvature applied to it. The final mesh is rendered using Runtime Mesh Component *[17]*. This allows us to render large amounts of dynamic data, as well as update the mesh on a separate thread when needed. In Figure 13 the final clipmap mesh is visible in wireframe.

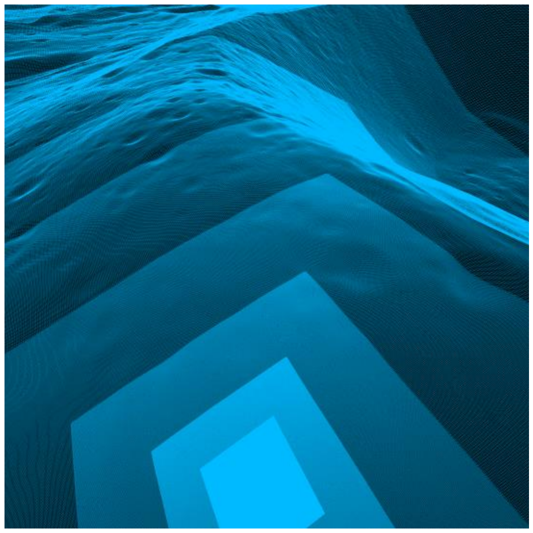


Figure 13. UE5 Clipmap Wireframe View

Heightmap Parsing— A unique feature of this terrain method is its ability to parse the DEMs at runtime and present a fixed size mesh regardless of the amount of lunar terrain data provided. This mesh updates the position and normal vector for each vertex after the camera has moved a set update distance. Default heightmaps are provided for approximately 4000 square kilometers of terrain at varying resolutions, or the user can substitute their own. Any number of DEMs can be imported, limited only by the system memory required to load them. The highest resolution DEM takes precedence over lower when coordinates overlap. When the vertex grid of the DEM does not align with the generated clipmap grid, we interpolate the height with the 4 nearest data points to get the height for the specified vertex. The normal vectors for each vertex are calculated using slopes obtained by sampling nearby points on the heightmap.

Clipmap Details—When the camera is moving, either in the Unreal Engine editor or in the running DUST application, the mesh updates at a rate calculated to lower the number of new vertex positions, which limits visual artifacts. This rate defines how far the camera needs to move before the clipmap terrain updates. This means that the rate cannot be larger than the first level of the clipmap, since the camera should not be able to get to an area with lower resolution. Technically the best update rate would equal our largest level's resolution because that would mean that each vertex would move into an existing vertex position, or between two existing vertex positions. However, if we used this rate, we would violate the rule where the rate must be smaller than the first level. Therefore, we use the largest level's resolution where the resolution is still smaller than the size of the first level. When the calculated update is reached, the mesh pulls vertex information from the stored heightmap data on a separate thread.

Clipmap Terrain Tradeoffs—The clipmap terrain method can generate lunar terrain for thousands of kilometer squared regions by directly reading heightmap data. This process ensures that there is no precision loss from the source data to the mesh construction and lunar curvature is incorporated as the data is applied to the clipmap mesh. Due to the clipmap's resolution fall-off nature and configurability, the render size and resolution can be scaled to meet hardware requirements without modifying the source data. This feature also ensures the method remains memory efficient as the majority of the memory usage results from the multiple gigabyte sized DEM files being loaded at runtime. If the DEM files are overloading a system's hardware requirements, the user can remove unnecessary heightmaps providing only the required data to the clipmap generator. Additionally, the lunar terrain clipmaps can be utilized on a variety of platforms including Windows, Mac, Linux, and Virtual Reality. Current limitations to the clipmap method include several possible visual artifacts. When traversing the terrain at a relatively fast camera speed visual artifacts can occur when the clipmap level resolution updates. Additionally, it is possible for shadows to be incomplete when a clipmap is rendered with low level counts that include partially formed terrain features. Finally, when abruptly transitioning from terrain locations with low resolution such as 15 mpp to much higher resolution such as 20 cmpp there may be noticeable resolution updates as the clipmap levels adjust.

Summary

Three methods for generating lunar terrain in Unreal Engine 5 have been developed each with unique benefits and limitations. The visual results from each method are similar with few noticeable differences. When choosing a terrain generation method, the decision is primarily utility driven. For example, the landscape method is the easiest to use, however, it has limitations on precision. Nanite terrain meshes provide the highest fidelity and accuracy, however, they have stringent hardware limitations. The method currently utilized in DUST is clipmap terrain generation due primarily to its scalability and platform flexibility. This method allows us to tailor the terrain's detail to meet hardware requirements as well as adapt the terrain as new data becomes available. Its limitations are primarily visual artifacts which can be mitigated by increasing computational resources. Overall, clipmap terrain generation utilizes novel technologies with trusted techniques to present a versatile real-time terrain rendering method.

LUNAR ENVIRONMENT MODELING

Lunar Curvature

Lunar curvature is easy to overlook in low-fidelity lunar terrain simulations as its inclusion drastically increases the difficulty of accurately positioning objects on the terrain's surface. Unreal Engine 5 utilizes a left-hand coordinate system with the x-axis forward, y-axis right, and z-axis up. Applying curvature to convert a flat plane to a sphere of radius 1,737,400 meters results in a modification to each point in the plane on all three axes.

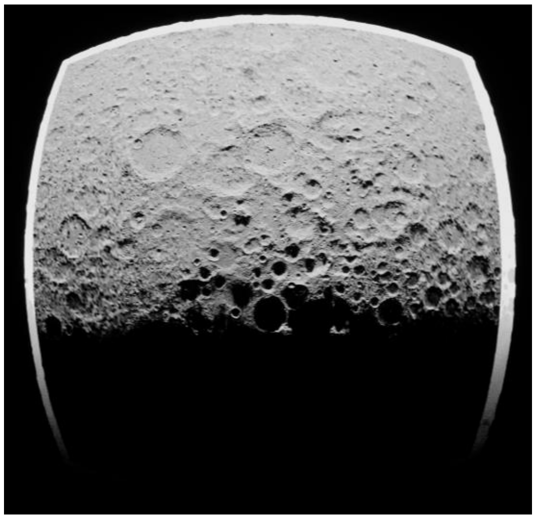


Figure 14. Lunar Curvature Representation

In DUST, when the positions are close to the South Pole the effect is minimal as this is where the origin of our coordinate system is located. However, as the terrain extends in the xy-plane, the magnitude of the offset applied by curvature to each point increases when compared to its uncurved original position. Therefore, when calculating positions in areas close to the origin the effect of curvature is minimal and can even be less than 1 m. This can lead to the false conclusion that curvature is insignificant in the generation of lunar terrain. However, when considering the cumulative effect of curvature over 100s of kilometers of lunar terrain the shadow lengths and world positions change significantly. The overall contribution of curvature applied to over a thousand kilometers in width of lunar terrain provides a visual distortion demonstrated in Figure 14.

Lunar Surface Texture

Our rendered terrain surface texture is a constantly evolving material that is continuously improved to increase fidelity and visual acuity. Its goal is to provide up-close fine grain texturing as well as accurately reflect the lunar surface's albedo at increased distances. In our initial attempts we encountered evident material tiling due to the varying resolution terrain causing the UV axis of our textures to scale inconsistently. To resolve this, we removed UV values from our terrain meshes and instead utilized our Unreal world positions as our texture's UV coordinates. This change provided us with identical texture response regardless of the mesh's resolution. To create texture tiling that provided details without obvious repetition at any height above the surface we needed to minimize apparent repetition by linearly interpolating sets of normal map textures with varying scale factors applied. These normal sets are swapped as the height above the surface increases or decreases. An additional problem we encountered was the extremely low light angle present at the lunar south pole. The low light angle caused our material's normal map to appear dark, washing out any visible details. We apply normal flattening at varying distances to maximize details depending on the camera's height above the surface. The lunar surface texture, as represented on the DUST lunar surface, is demonstrated in Figure 15.

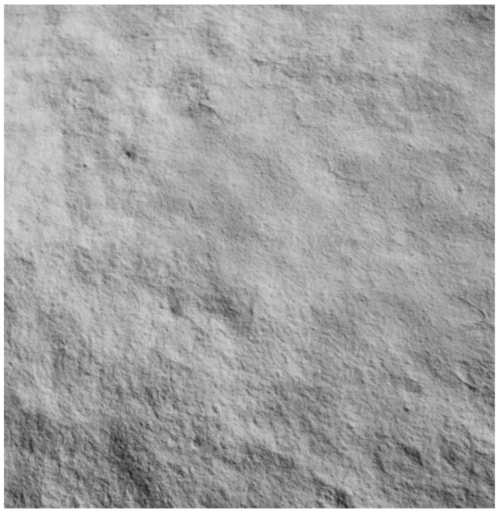


Figure 15. Lunar Surface Texture

Rocks and Craters

To increase the realism of the lunar terrain, additional features such as rocks and craters are added. The positions and scales of these features are statistically representative of the lunar south pole and captured within PODB. This information is used to apply these features to the DUST scene. The craters are distributed via an upscaled DEM, which currently achieves a maximum resolution of 20 centimeters per pixel. This is accomplished by upscaling the 5 meters per pixel LRO data and overlaying the crater distributions directly on the DEM. This process is outlined in DLES [1]. The DEM can then be read into our clipmap terrain. This capability allows DUST to generate craters anywhere on the lunar surface where a 20 centimeters per pixel heightmap file has been provided. The lunar rocks are generated at runtime by reading a JSON file received from PODB. The JSON file contains the coordinates, scale, and model type for each rock instance. In Figure 16 both the rocks and craters are visible on the lunar surface.

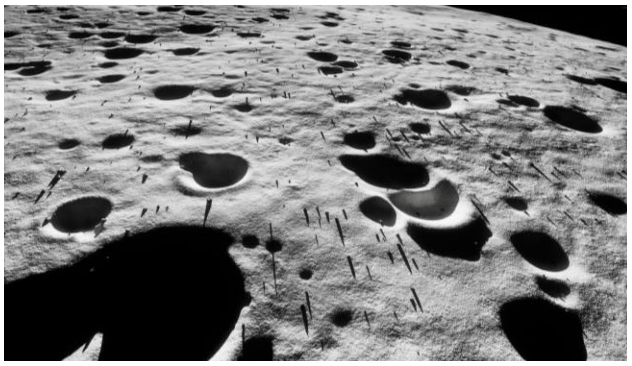


Figure 16. DUST Rocks and Craters

The rocks are placed in a circular area around a specified latitude, longitude, and radius. The meshes utilized to create the lunar rocks are 3D scanned digital twins of collected Apollo era rock samples. The scanned samples are scaled to provide additional variation in the rock layers. Currently, all rocks are generated and displayed in the scene at once with no partitioning applied to improve performance. Rock caching/partitioning is not necessary at this time due to the application of Nanite to all rock meshes used in the lunar scene. Nanite provides highly detailed geometry that has a minimal impact on performance. Additionally, it determines which mesh details can be reduced depending on the camera position and adjusts the level of detail to optimize performance without changing visuals [16]. Therefore, we are capable of rendering hundreds of thousands of rocks with little to no decrease in frame rate. Additionally, all rocks are rendered with collisions and because they are generated from PODB data they match across our various rendering technologies.

Lumen

Lumen is a new feature in Unreal Engine 5 which provides dynamic bounce lighting approximations. Lumen can provide bounce lighting with either software ray tracing or hardware ray tracing [18]. Figure 17 demonstrates how Lumen can render bounce lighting off a surface. Hardware ray tracing provides the best results at the highest performance cost and is limited to Windows OS with DirectX 12 [18]. In our DUST instance we utilized Lumen with software ray tracing to provide bounce lighting inside craters, within rock shadows, and off objects like rovers and astronaut extravehicular activity suits. The dynamic lighting helps illustrate crater depth and the impact of the sun's intensity without an atmosphere to hinder it.

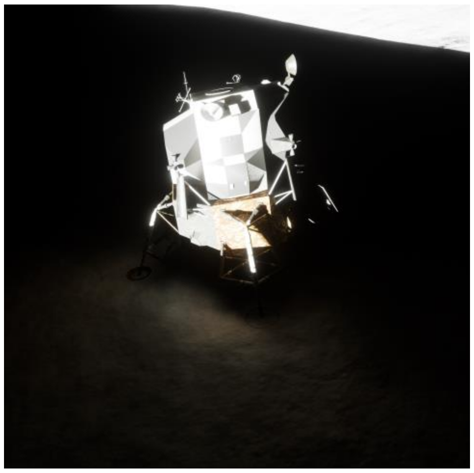


Figure 17. Lumen Lighting

Virtual Shadow Maps

Virtual Shadow Maps (VSMs) are a novel shadow rendering method in Unreal Engine that employ high resolution shadow maps of 16k-by-16k pixels supplemented with clipmaps to increase resolution for directional lights in large worlds [19]. VSMs are optimized for Nanite geometry and provide a dynamic shadow rendering solution capable of displaying detailed shadows over thousands of kilometers of terrain. Figure 18 is a rendering of around 75 kilometers of terrain in width, with high-fidelity shadows throughout. Additionally, there is minimal reduction in visible shadow accuracy as the distance to the mesh increases. With VSMs DUST can provide performant dynamic shadows for hundreds of thousands of Nanite enabled rocks and across craters with shadows spanning tens of kilometers in length. The shadows are also able to render dynamically as SPICE updates the position of the sun.

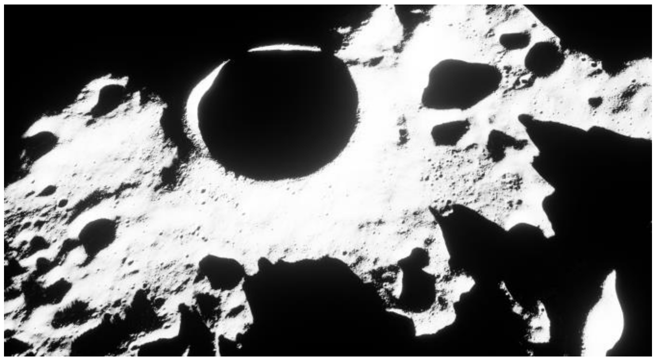


Figure 18. Shadow Rendering

Double Precision

Unreal Engine 5 now provides double precision in most areas of the engine. This means that instead of FLOAT32 as the primary data type of various engine components such as actor transforms the engine now utilizes FLOAT64, increasing the available precision. SPICE can now position the various planetary bodies such as the Sun and the Earth in their actual calculated positions without significant loss of precision. Previous implementations required scaling of the positions via a consistent scale factor to maintain accuracy. Additionally, we are now able to create simulations that involve spacecraft such as lunar landers, which may need to traverse distances of thousands of kilometers, with minimal precision loss.

FUTURE WORK

Our future work includes continuing to expand upon the existing tools and features in DUST, as well as experimenting with the inclusion of the DUST terrain into additional projects. As the newly released features in Unreal Engine 5 receive improvements and optimizations we hope to provide increased performance and a greater variety of supported platforms.

We are constantly improving the fidelity of DUST to increase its effectiveness in visualizing lunar terrain. One example is the current rock distribution capability which is being developed to provide a greater generation radius and increased fidelity for the smaller rock layers. PODB provides the statistical distribution for our rocks and craters currently in the DUST scene, however we are limited by the number of rocks we can add on specific platforms. While Nanite does provide us the capability to generate thousands of rocks with minimal performance impact, this feature is only available on machines running Windows OS. By incorporating the combination of Nanite and a rock grid caching system we hope to support rock generation over a 100 km by 100 km area of lunar terrain. Additionally, we are working to add to the feature set present in DUST. The traverse tool is currently limited to presenting primarily visual data with little empirical value other than time estimates. The tool is planned to deliver power and temperature estimations throughout the duration of a traverse at different times/dates. The fidelity of the communication analysis tool will be refined to incorporate radio transmission fall-off instead of an entirely line of site implementation. New features such as in-engine physics rover control, and lunar terrain impact deformation are undergoing preliminary development.

One platform we are beginning to implement Unreal lunar terrain in is virtual reality. Currently Unreal Engine does not support the use of Lumen or Nanite in virtual reality, however, the engine does provide multiple infinite light sources and double precision. These features would allow us to create high-fidelity virtual reality simulations utilizing the same clipmap terrain present in DUST. An additional future innovation is the use of DUST with Unreal Engine's nDisplay feature. The nDisplay feature in Unreal allows the user to map their Unreal instance to any number of rendering displays. We are hoping to implement this feature with a rover traverse simulation which visualizes the terrain via a wall of monitors each displaying a portion of a mapped DUST instance. Finally, we are creating a Pixel Streaming implementation to serve as an additional DUST distribution option. Pixel Streaming is another Unreal Engine feature that provides a framework for interacting with the Unreal instance via a web browser. This capability would allow us to distribute DUST to a wide array of platforms without hardware and performance limitations.

CONCLUSION

The DLES Unreal Simulation Tool is an advanced lunar visualization environment that presents unique tools and capabilities for high fidelity analysis of the lighting conditions and terrain details present on the lunar surface.

DUST's current platform support includes Mac, Linux, and Windows. Certain Unreal features such as Nanite are not available on all platforms. DUST includes a variety of tools to provide additional analysis capability including traverse rendering, communication analysis, heat map overlay, and slope and elevation map visualization. With the clipmap lunar terrain, we can adjust fidelity to match specific hardware requirements. Additionally, as new terrain data is made available the generation can be updated by simply replacing the stored heightmap files.

Novel Unreal Engine 5 features such as Lumen, Nanite, Virtual Shadow Maps, and double precision provide capability to render increased detail and scale. The addition of performant bounce lighting provided by Lumen demonstrates the effect of the Sun's intensity reflecting off various scene elements. Nanite allows in-simulation meshes to hold high vertex counts with little impact on performance, which is currently utilized on lunar rock rendering. Virtual Shadow Maps provide high resolution shadows over expanded regions of terrain that maintain visible accuracy as the camera's distance increases. The addition of double precision allows us to accurately position planetary bodies and incorporate engineering-based simulations with TRICK. Overall, Unreal Engine 5 presents a unique feature set that indicates promising initial results in its ability to support accurate simulation development.

DUST is currently in early stages of development and has not been widely distributed for use. Its designed purpose is to provide lighting and terrain analysis in support of future Artemis missions.

ACKNOWLEDGEMENTS

The authors would like to acknowledge and thank Paul Bielski, Zack Crues, and Eddie Paddock for your vision, guidance, and support in the development of DUST and in providing the DLES product suite. We would also like to acknowledge Steve Munday (head of the Human Landing Systems Crew Compartment Office) and Doug Craig (head of the Human Exploration Office, Systems Engineering and Integration, Strategy and Architecture Team) for the funding and direction.

REFERENCES

[1] E. Z. Crues, S. J. Lawrence, A. Britton, P. Bielski, J. Schlueter, A. Jagge, C. Foreman, C. Raymond and N. Davis, "Digital Lunar Exploration Sites (DLES)," in *2022 IEEE Aerospace Conference (AERO)*, 2022.

[2] "Moon Trek," NASA, [Online]. Available: https://trek.nasa.gov/moon/. [Accessed 09 2022].

[3] N. M. Estes, C. D.Hanger, A. A. Licht and E. Bowman-Cisneros, "Lunaserv Web Map Service: History Implementation Details Development and Uses," in *44th Lunar and Planetary Science Conference*, 2013.

[4] A. D. Garcia, J. Schlueter and E. Paddock, "Training Astronauts using Hardware-in-the-Loop Simulations and Virtual Reality," in *AIAA Scitech 2020 Forum*, Orlando, 2020.

[5] "Scientific Visual Studio," NASA, [Online]. Available: https://svs.gsfc.nasa.gov/. [Accessed 6 September 2022].

[6] NASA, "NASA's Lunar Exploration Program Overview," September 2020. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/artemis_plan-20200921.pdf. [Accessed September 2022].

[7] "MaxQ: Spaceflight Toolkit For Unreal Engine 5," Gamergenic, 25 12 2021. [Online]. Available: https://www.gamergenic.com/project/maxq/. [Accessed 2022].

[8] "Trick Simulation Development Environment," NASA, August. [Online]. Available: https://nasa.github.io/trick/. [Accessed September 2022].

[9] J. M. Penn and A. S. Lin, "The Trick Simulation Toolkit: A NASA/Open source Framework for Runnning Time Based Physics Models," in *AIAA Modeling and Simulation Technologies Conference*, 2016.

[10] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," *ACM Transactions on Graphics*, vol. 23, no. 3, p. 769–776, 2004.

[11] R. Campos, J. Quintana, R. Garcia, T. Schmitt, G. Spoelstra and D. M. A. Schaap, "3D Simplification Methods and Large Scale Terrain Tiling," *Remote Sensing*, vol. 12, no. 3, p. 437, January 2020.

[12] United States Geological Survey, "What is a digital elevation model (DEM)?," [Online]. Available: https://www.usgs.gov/faqs/what-digital-elevation-model-dem. [Accessed 09 2022].

[13] G. Contributors, "GDAL/OGR Geospatial Data Abstraction Software Library," Open Source Geospatial Foundation, 2021. [Online]. Available: https://gdal.org. [Accessed 15 08 2022].

[14] "UnrealGDAL: Unreal Engine GDAL plugin," TensorWorks Pty Ltd., 2020. [Online]. Available: https://github.com/TensorWorks/UnrealGDAL. [Accessed 2022].

[15] "Landscape Technical Guide," Epic Games, [Online]. Available: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/Landscape/TechnicalGuide/. [Accessed 2022].

[16] "Nanite Virtualized Geometry," Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine. [Accessed 2022].

[17] TriAxis-Games, "Runtime Mesh Component Git Hub," [Online]. Available: https://github.com/TriAxis-Games/RuntimeMeshComponent. [Accessed 09 2022].

[18] "Lumen Technical Details," Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/lumen-technical-details-in-unreal-engine/. [Accessed 09 2022].

[19] "Virtual Shadow Maps," Epic Games, [Online]. Available: https://docs.unrealengine.com/5.0/en-US/virtual-shadow-maps-in-unreal-engine/. [Accessed 9 2022].

BIOGRAPHY

Lee Bingham *received a B.S. in Computer Engineering from the University of Houston – Clear Lake and a M.S. in Electrical Engineering from the University of Houston. Lee began his career at NASA Johnson Space Center in 2016 with the Software, Robotics and Simulation Division and has contributed to simulation and visualization development for the NASA Exploration Systems Simulations (NExSyS) team and the Virtual Reality Training Lab. He is the lead of the Digital Lunar Exploration Sites Unreal Simulation Tool (DUST) and the Lunar Surface Mixed-Reality and Active Response Gravity Offload System (ARGOS) projects.*

Jack Kincaid *received a B.S. in Computer Engineering from Purdue University in 2021. He is a Software and Simulation Engineer for METECS supporting NASA Johnson Space Center's Software, Robotics and Simulation Division under the Simulation and Graphics Branch. He has several years of experience in game engine development with a recent focus in Aerospace applications. Jack is one of the lead developers of the Digital Lunar Exploration Sites Unreal Simulation Tool (DUST).*

Benjamin Weno *received his B.S. in Computer Engineering from Iowa State University in 2018. He is a Software and Simulation Engineer for METECS supporting NASA Johnson Space Center's Software, Robotics and Simulation Division under the Simulation and Graphics Branch. His professional experience includes full stack and game engine development. He is currently a lead developer on the Digital Lunar Exploration Sites Unreal Simulation Tool (DUST).*

Nicholas Davis *received a A.S. degree in Physics from San Jacinto College in 2010. Through his fifteen-year career in software development, he cultivated expertise focusing on modern web-based architectures and high availability data models. Leveraging his sharp acumen in this area, Mr. Davis developed web-based solutions for the NFL providing valuable situational insights during key events. He has also developed core logistical technologies used by telecom networks as well as oil and gas providers. Currently, Mr. Davis designs intersystem engines for the NASA Prototype Immersive Technology (PIT) lab, enabling a network of VR elements to function cohesively in lunar exploration simulations.*

Eddie Paddock - *Virtual Reality (VR) Technical Discipline Lead (TDL) for NASA's Engineering Directorate at the Johnson Space Center (JSC), Houston, TX. Over 40 years of experience in developing and managing aerospace and robotics simulation and graphics projects including most recently eXtended Reality (XR) applications for ISS, Gateway and Artemis Programs.*

Cory Foreman *received B.S. and M.S degrees in Aerospace Engineering from Iowa State University in 2006 and 2008 respectively. Upon completion of his M.S., Cory began his career at NASA Johnson Space Center in Houston, TX where he has contributed to the design and development of the Orion crew capsule and Training Systems 21st Century (TS21). Cory has over 13 years of experience in simulation and software development and currently works as a simulation and* *software engineer in the Simulation and Graphics Branch where he contributes to development of simulations focused on lunar landing and surface exploration systems.*