Accurate Planetary Mesh Generation in Full-Scale Simulation Environments

Dillon Fisher, Michael McCarthy, Connor Jakubik, Patrick Zhong, Wade Smonko,
Kendall Mares, and Gregory Chamitoff, *Member, IEEE**

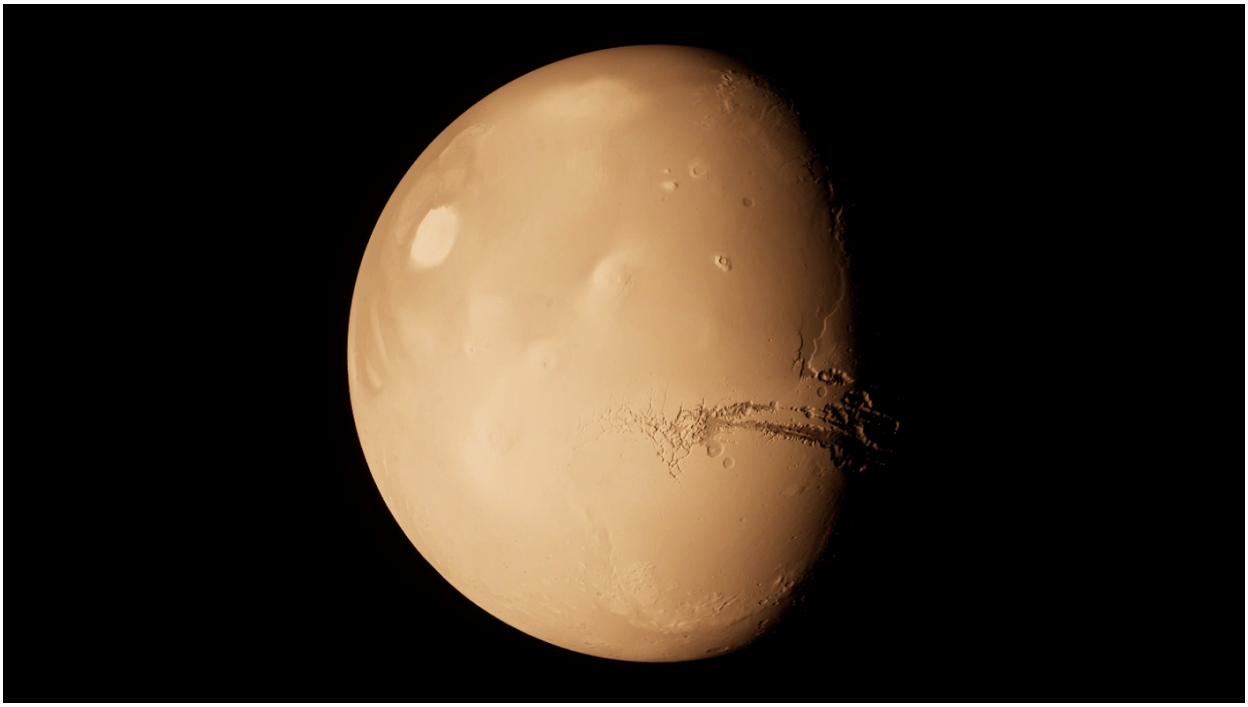

Fig. 1. The planet Mars rendered to scale in Unreal Engine 4 using our Planetary Mesh Generation Technique.

Abstract—Procedurally generated terrain is widely used in the gaming industry, and has seen increased use for engineering VR simulations in recent years. These terrain generation methods allow for rapid development of high fidelity environments, and since the generation is seeded in coherent noise functions, require very little storage space. A simulation requiring high fidelity height maps for known celestial bodies like the Moon or Mars, cannot solely use procedural terrain. This paper presents a system for generating surface meshes through the combination of a Digital Elevation Model (DEM) and procedural terrain techniques, which allows a planet to have scientifically defined macroscopic detail, while leaving finer levels of detail (LODs) to be determined by Perlin noise functions. This provides a recognizable topography for a rendered planet as viewed from orbit, while also retaining realistic, procedurally generated terrain on the surface. Using a quadtree data structure, we can optimize the rendering of the planet at multiple levels of detail, dynamically displaying higher levels of detail around the user and minimal detail at points of the surface far from the user. Typical planetary DEMs do not have high surface spatial resolution. The Lunar Orbiter Laser Altimeter (LOLA), for example, has collected height data from the Lunar surface, generating a full DEM with a surface spatial resolution of 118 meters. The system presented here is designed to display measured height data to the user until it reaches a level of detail beyond the original DEM. At this point, the system interpolates between those data points with procedural terrain. This system has been used to render most of the inner planets in our solar system and the Moon at their proper scales using the DEMs provided from the United States Geological Survey (USGS). The system is lightweight enough to be used in a larger VR simulation, with both surface and orbital operations.

Index Terms—virtual reality, planet modeling, terrain generation, DEM, quadtree

1 Introduction

Terrain mesh generation is a necessity for most video games today and is becoming increasingly important for engineering simulation environments. In the past, games could afford to artistically create a small landscape mesh in a scene, but as the environments which games encompass have grown in scale, it becomes unreasonably difficult to manually sculpt these huge environments with enormous polygon counts. The task becomes especially difficult when these environments need to represent fully realized worlds, with expansive but still highly detailed regions. Due to this difficulty, the industry has been steadily moving towards procedurally generated height maps to vastly increase

• *Texas A&M University. E-mail: dillon.fisher.14@tamu.edu, michaelcmcc1@tamu.edu, connor.jakubik@gmail.com, patrickzhong@tamu.edu, wsmonko@hotmail.com, k_mares16@tamu.edu, chamitoff@tamu.edu.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

the rate of terrain design/creation. Games that do this typically design a system or algorithm to mimic natural landscapes, such as deserts, mountain ranges, or even the Moon's surface. Most of these meshes do not contain real terrain data, but are procedurally generated to look close to a desert or lunar landscape that they represent.

This paper presents a system for rendering the surface meshes of various celestial bodies at runtime, with dynamically varying levels of detail. This system combines the data collected by satellites in the form of Digital Elevation Models (DEM) and the use of modern procedural generation techniques using Perlin noise to achieve scientifically defined macroscopic detail from orbital perspectives while also having realistic, procedurally generated detail on the surface.

2 Purpose

This system was designed as a component of an engineering design platform called "SpaceCRAFT" to meet the needs of high-fidelity space mission simulations, often using VR technology. SpaceCRAFT is a collaborative research and analysis tool primarily focused on space exploration simulations, with an emphasis on VR. It consists of a server for high-speed, asynchronous computation, and clients for visualization and human-in-the-loop interaction. In order to have representative terrain for planet surfaces in these simulations, we require plausible detail down to human scale. To achieve this, we need to use the highest resolution terrain data that is available from satellite measurements, and augment the data to achieve higher detail. There are some terrain rendering tools already available in Unreal Engine 4, the game engine SpaceCRAFT utilizes, but those are constrained to flat terrains with finite areas, and cannot easily be changed with new data once the program is compiled. Facing these limitations we opted to make our own procedural planet surface mesh generation tools that are capable of accepting terrain data at any resolution, reducing it to optimized data structures, and rendering without artifacts at high speeds for VR viewing (typically 90 frames per second).

There are several challenges in meeting the diverse needs of real-time VR simulations, especially for space exploration simulations. Consider, for example, a VR simulation of a crewed Entry, Descent, and Landing (EDL) mission sequence, possibly for crew training purposes. To have representative visuals, the planetary surface will need appropriate detail at all stages of the mission, from orbit to the surface. This, and many other space simulations present an issue of scale, especially when rendering celestial bodies such as planets. This application requires real-time management of surface detail with respect to the user's position, while also accounting for the floating point error introduced by such a large environment. Maximum surface detail is constrained by the amount of data available on a particular celestial body. If this EDL simulation were to take place on Mars, for example, even some of the most detailed global datasets available have a spatial resolution of about 200 meters [3]. This is inadequate for a meaningful Mars surface simulation. Our system was designed to solve these issues, offering real-time surface detail management, mitigation of floating point error artifacts, and the ability to augment planetary datasets to achieve highly detailed surface simulations.

3 Related Works

Generation of fully procedural planets is widely used in the game industry, and has been for many years. Perlin Noise, which is the most commonly used coherent noise algorithm for terrain generation, was created in 1983 by Ken Perlin for Disney's Tron. Perlin went on to publish his work in 1985, inciting huge growth in the procedural texture generation field [7]. Since then, there have been dozens of papers on the generation of terrain. By 2001, Sean O'Neal [6] had written a paper on mesh generation of spherical planets using a dynamic level of detail algorithm. His algorithm and many others since have utilized six separate trees to generate the triangles on the surface of the planet. These trees begin as the six faces of a cube, with two triangles each. As the level of detail increased, the triangles split into smaller triangles, progressively transforming the cube into a sphere and further into a planet.

Livny [4] expanded on the mesh generation algorithm by creating a quadtree system that breaks the surface into small patches, where each patch is a fixed level of detail and can increase and decrease in detail independently. Similarly, d'Oliveira [2] used six of these patch, level-of-detail systems to generate a full planet. d'Oliveira would interpolate the values between these six faces of the planet to ensure the continuity between them.

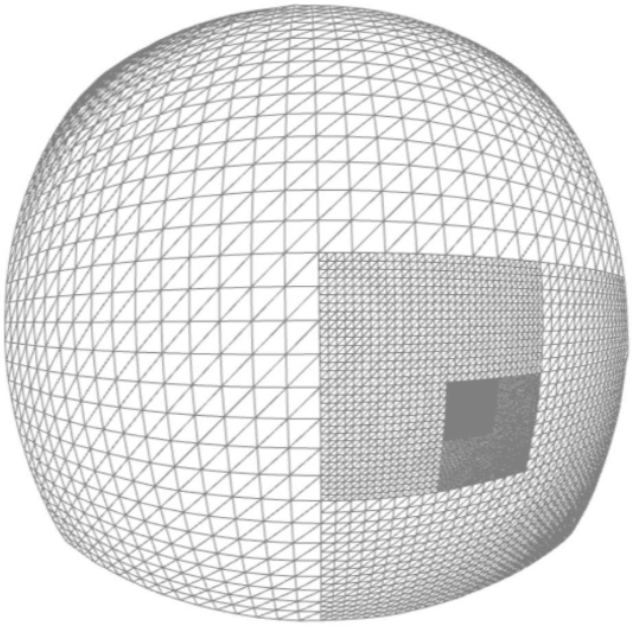

Fig. 2. A sample planetary surface quadtree displaying high LOD in localized regions.

These planetary generation tools create fully procedural planets that imitate terrain on known celestial bodies, but do not accurately reproduce them. Livny's [4] algorithm samples from a digital elevation model (DEM), allowing real terrain data to be rendered at multiple resolutions, but this system is most beneficial when used with highly detailed terrain data. Unfortunately, the highest spatial resolution available for planetary datasets is typically hundreds of meters. Therefore, this system would not be fully utilized if it were used to render a planet at full scale, unless the detail of the meshes were increased beyond that of the original DEM.

4 System Overview

Our system generates meshes from planetary DEMs while employing Perlin noise functions for finer surface detail. Similar to Livny [4], we use a quadtree to dynamically generate meshes with a fixed number of vertices at a resolution of 65x65. These meshes initially sample the supplied DEMs to create low detailed polygons across the entire planet. When more detail is needed, each mesh individually generates four smaller meshes with the same number of vertices, quadrupling the height detail displayed in that area. Eventually the level of detail of the meshes will reach that of the original DEM. At this point, the meshes will upsample the original dataset, filling in the gaps with linearly interpolated height data plus a Perlin noise offset to achieve surface detail where no data exists. Utilizing a noise function of an appropriate frequency, we can dynamically generate surface detail, while upholding the accuracy of the original dataset. Our system has three primary components: preparing the global height maps to be quickly processed at runtime, noise interpolation at higher levels of detail, and implementation of a mesh-quadtree level of detail algorithm to dynamically control the number of polygons at different places on the surface. These three components allow us to generate a planet with a high level of accuracy and realism from both orbital and surface viewpoints.

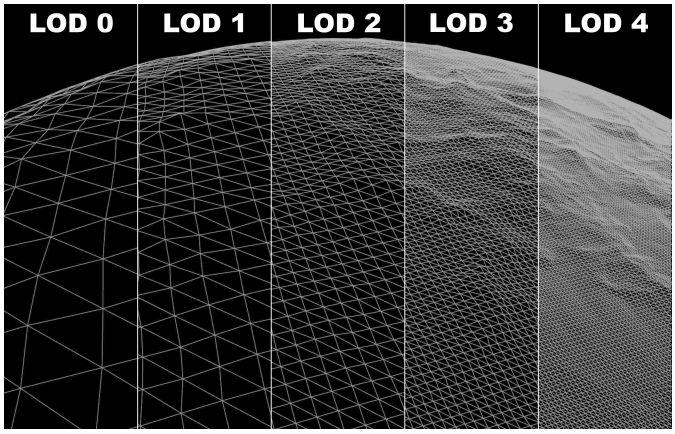

Fig. 3. A wireframe rendering of the first 5 quadtree levels of detail, for a fictional planetary surface.

4.1 Quadtree LOD Algorithm

As mentioned above, we use a quadtree level of detail system to dynamically generate the planetary surface meshes. A typical global DEM is gigabytes in size, so it is not ideal to render the planet in its entirety as the excess of polygons will adversely affect performance. The quadtree allows us to segment different areas of the planet and render high detail only where it is needed. For our simulations, we use the user location relative to each of the meshes in the quadtree to determine where meshes should split to increase the level of detail. The goal is to have a high enough spatial resolution around the user to produce realistic terrain while minimizing the number of meshes out of view, and therefore the performance impact.

The quadtree implementation that we designed has a mesh at each node. Each mesh can then be split into four smaller meshes, each with the same number of vertices. When this happens the four new meshes take up the same surface area on the planet as the single mesh before, but with quadruple the original number of polygons, effectively increasing the visible surface detail in that area. Each of these four new meshes can then individually split further, increasing the detail in an even smaller subsection of the planet. These meshes can hypothetically continue to split indefinitely, displaying higher and higher resolution of detail until the computer runs out of memory. To prevent this, we cap the number of splits of the quadtree as a function of the radius of the planet. This not only prevents the mesh data from overflowing the memory, but also standardizes the expected surface spatial resolution across planets of varying sizes.

4.2 Mesh Global Positioning

Similar to O'Neal's [6] planetary mesh generation algorithm, ours uses a cube as the starting point to generate spherical bodies. Our planet generation process uses a cube representation for both the data in the supplied DEM and the sampling of the Perlin noise functions, where the 6 faces of the cube corresponds to the 6 separate quadtrees described above. Without any quadtree mesh splitting, each face of the cube is a single mesh with a fixed number of vertices. Each mesh can be interpreted as a 65x65 two dimensional array, with equally spaced x and y coordinates that act as indices. The z coordinate for a given x and y index, stores the surface height variation sampled from data and procedural techniques.

Figure 4 shows the relationship between this cube representation of the data and a cube map projection, used for importing planetary surface data. The local coordinate system for each face is also shown, which allows for a mapping between the two representations that takes care of any transformations. The origin of each face coordinate system can be represented in the global coordinate system also shown.

Our system manages the position of meshes using the global coordinate system. When the quadtree system splits the meshes, the coordinates of the new meshes are determined first in the local two

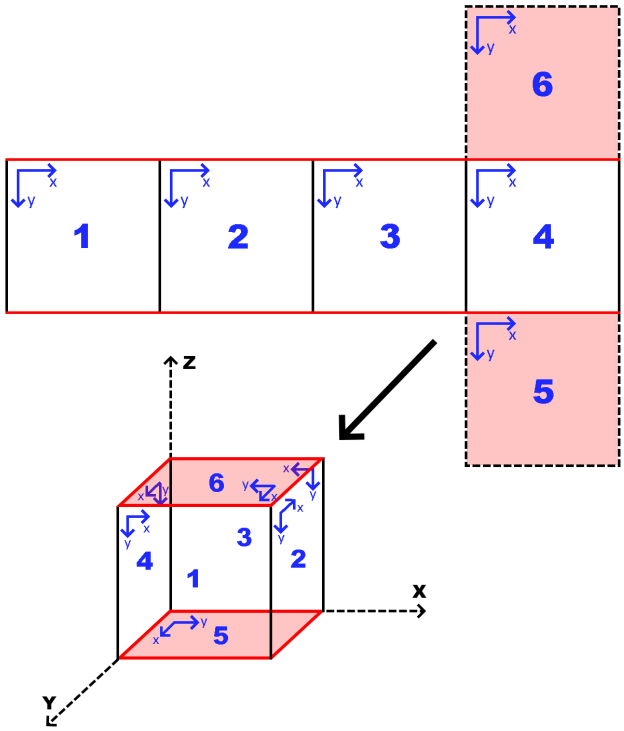

Fig. 4. The transformation of local coordinates for each cubemap face, starting from the flattened cubemap projection.

dimensional coordinate system of their face, and then transformed into the global coordinate system for rendering. Each mesh also has its own local coordinate system to manage the position of each of its vertices. The coordinate systems in use can be summarized as:

$$Frame\ G = Global\ coordinate\ frame\ (Unreal\ Engine)$$

$$Frame\ F_n = Face\ coordinate\ frame\ for\ n^{th}\ face$$

$$Frame\ M_{i,j,k} = Mesh\ coordinate\ frame\ for\ a\ mesh\ at\ the\ (i^{th}, j^{th})$$

$$position\ at\ the\ k^{th}\ level\ of\ detail\ on\ a\ face$$

In general, coordinates expressed in the face and mesh coordinate systems will require a simple transformation to be expressed in the global coordinate system. The position of any particular mesh coordinate frame in terms of global coordinates can therefore be expressed as a simple sum of position vectors, where T_{AB} is the position of A with respect to B, in global coordinates.

$$T_{M_{ij,k}G} = T_{F_nG} * T_{M_{ij,k}F_n}$$

This coordinate convention allows for the global positioning and aligning of each mesh with its neighbors. These coordinate transformations also enable each mesh to correctly sample the DEM data and Perlin noise functions to determine the height of each vertex.

4.3 Cube to Sphere Warping

The cube representation described so far is only a starting point for the creation of spherical bodies. In the process of projecting celestial height data onto the surface, the cube representation is warped into a spherical projection. In similar frame and position vector notation, the position of any particular vertex p can be represented in global coordinates as the following vector sum.

$$P_G = T_{F_nG} * T_{M_{ij,k}F_n} * P_{M_{ij,k}}$$

This relationship holds before and after spherical warping, allowing a consistent transformation between local and global coordinates.

To warp the surface of the cube representation into a sphere, a transformation is applied to each vertex along the cube. The transformation can be represented by the following expression.

$$P_{G,sphere} = (R_{planet} + h) * \hat{P}_{G,cube}$$

$$R_{planet} = the\ planet\ radius$$

$$h = the\ height\ of\ the\ surface\ variation$$

$$\hat{P}_{G,cube} = the\ original\ unit\ direction\ vector\ to\ the\ vertex$$

The surface variation h in the previous expression is where all the surface terrain variation is defined, with height data coming from both real planetary data and noise interpolation techniques. For a particular vertex, h is retrieved through sampling data or continuous noise functions, in which the coordinates of the vertex determine the returned value of h. This system configuration separates the mesh positioning and quadtree system from the specific surface terrain being rendered. The planet radius affects only the tangential spacing between surface vertices, while the radial component is allowed to vary according to the height of the surface variation.

4.4 Asynchronous Mesh Generation

For applications like engineering simulations, our dynamic LOD system will run continuously in order to bring planetary meshes to a desired LOD. Since the splitting and merging of these meshes is on demand, it happens with little warning and can cause sharp dips in performance when higher levels of detail meshes are needed. Left alone, the result would cause frequent hitches and could periodically freeze the simulation's other processes, such as engineering models. Most of the heavy computation lies in sampling the noise functions, warping coordinates, and calculating vertex positions and normals on the fly, rather than intensive rendering operations. Since each splitting and merging operation for a mesh is completely independent of another, they are suitable operations for parallel processes. We thus mitigated the frame-rate hitches by deferring the computations, such as computing vertex position and normals of a new mesh in a background thread, until they are finished and can be brought back to the main thread for rendering.

We use a semaphore to keep track of the number of meshes that are still generating or deleting, in order to properly clean up after each mesh's split or merge operation. For example, when a quadtree splits, the original mesh is not immediately deleted to prevent flickering, which is the absence of any mesh in an area for a fraction of a second. The four new mesh generation tasks are sent to the thread pool, each incrementing the semaphore and generating the new mesh to be rendered. When all four new meshes are generated, the original mesh is deleted, leaving the four higher detailed meshes in its place. Similarly, for mesh merging, the four higher detailed children meshes wait until the new parent mesh is created before they are deleted. As a result, a brief moment of overlap occurs between the original and split meshes, but is far less noticeable than the flickering or the performance hitches and spikes that would result from synchronous generation.

4.5 Preprocessing Height Data

For our use case, we primarily work with known planetary bodies. The United States Geological Survey (USGS) distributes cylindrical projection of Digital Elevation Models (DEMs) of the moon and the four inner planets. It would be convenient to keep it in the given cylindrical format, but would require us to project the data to a cube map at runtime to integrate well with our terrain generation system. In order to lower overhead at runtime, we converted and stored the provided cylindrical projections of the planets as cubical projections in six smaller files. Figures 5 and 6 demonstrates the cylindrical to spherical conversion using the Moon DEM. This allows the data to be quickly read and the height maps generated in a way ideal for the quadtree. Each of the cubical projection files correspond to a different quadtree.

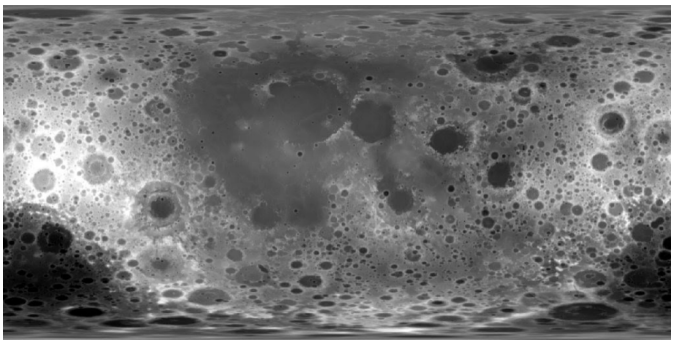

Fig. 5. The DEM of the Moon supplied by the USGS based on data gathered from the Lunar Orbital Laser Altimeter (LOLA)

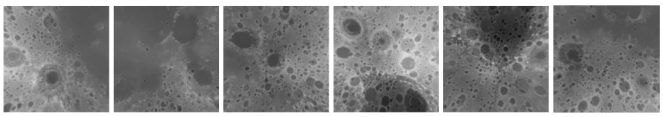

Fig. 6. The DEM of the Moon split into six faces of a cube map as represented in our digital isometric landscape (.dil) files

DEMs from the USGS database come in a variety of formats. Generally, they are stored as PNGs and TIFFs, but there is no common resolution or bytes per pixel for these files. To standardize these datasets for our use, we designed the Digital Isometric Landscape (DIL) file. This file type was developed to accelerate the loading process of the DEM datasets at runtime by storing the data in a way that is easily processed by the quadtrees. These files contain a three byte header; the first two store the 16-bit resolution of the image and the third byte stores the number of bytes per pixel of the image. The rest of the file is pure height data in row column order, following the x, y coordinate convention shown in Figure 4. To generate these files, we perform a cubic projection on the cylindrical height maps provided by USGS, splitting the original global DEM into the six DILs that our system can read. Each pixel of the DIL image corresponds to the height of a vertex on a given planet, with respect to the mean radius. At runtime, sampled data points will be read and loaded into a mesh.

The DILs have specifically defined characteristics allowing them to easily integrate into our system and facilitate quick access at runtime. Firstly, the surface ordering and coordinate system for each face follow our defined convention shown in Figure 4. Otherwise the six faces of the planet would not seam together and the rendered planet would look like six unaligned puzzle pieces. Secondly, the outer pixels of the DILs match the edge pixels of the adjacent DILs. This is due to the fact that each pixel represents a vertex on the mesh and therefore along the face seams of the planet the vertices are drawn in two separate quadtrees. In order for that vertex to have the same height position in both quadtrees, they must have the same value in the corresponding DILs. Finally, not only does the DIL have a square resolution, but the resolution is a power of two plus one.

$$DIL\ resolution = 2^n + 1\ for\ any\ integer\ n$$

This is to ensure that when the quadtree splits into four higher resolution meshes, the vertices of the four new meshes also align with the DIL's data.

4.6 Noise Interpolation

After the DIL files have exhausted their available spatial resolution, the quadtree will continue to split to finer levels of detail, interpolating the height data points from the DEMs with Perlin noise. Perlin noise works by randomizing unit gradients in an n-dimensional grid and then interpolating values between the points in the grid to form a smooth and continuous n-dimensional function. Noise functions are the result of summing one or more Perlin noise layers to produce a compound

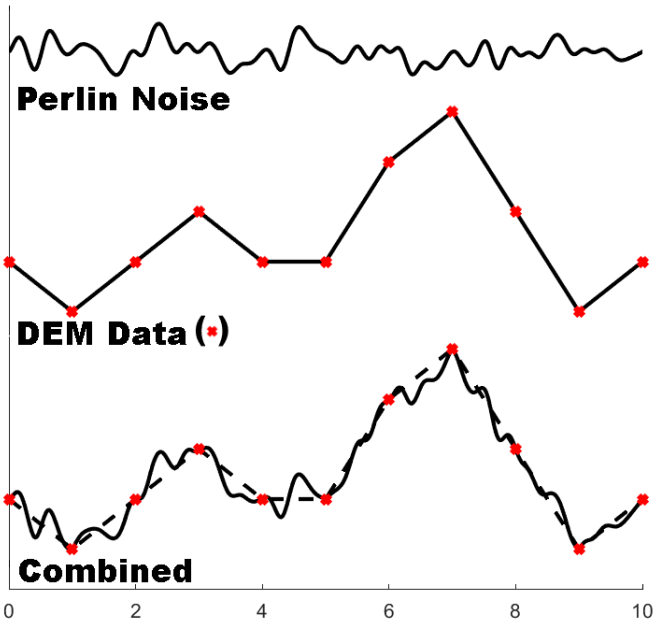

Fig. 7. A 2D example of the noise interpolation process. DEM data is augmented to achieve highly detailed surfaces, while leaving the true data points unaltered.

function. In these compound functions, each additional layer of noise has higher grid resolution than the last, which is referred to as an octave. Increasing the number of octaves for a noise function effectively adds more complex detail at finer spatial resolutions. One of the characteristics of Perlin noise is that the return value of the gradient interpolation function approaches zero at each of the grid points. While this may seem like an unwanted feature in a noise function as it produces a recognizable pattern, it is exactly what we need to ensure that all the real data points remain unaltered.

Using Libnoise, an open source Perlin noise library that allows for easy manipulation of Perlin noise layers, we designed a three dimensional Perlin noise function to generate realistic terrain detail in between the known DEM data points [1]. The procedural terrain generator created by d'Oliveira [2], and many others, create six separate height maps for their planet and then interpolate between them along the mesh edges. Since we are using a three dimensional noise function, we sample the Perlin noise function along the surface of a cube, where each face of the cube corresponds to a different mesh and DIL file. This sampling allows our six procedurally generated faces to naturally converge to each other along the edges since the edges are sampled from the same points in the noise function, removing the need to interpolate between the faces.

We also designed our Perlin noise function such that the resolution of the gradient grid at the lowest octave lines up with the highest resolution of the DIL files. Since the grid locations of the lowest octave line up with the height data from the DIL, we can sample the noise at any point along the surface of the cube and add the returned value to the linearly interpolated height from the dataset. When the point is a known height value from the DIL files, it has an integer location in the Perlin noise function grid, consequently, the function will return zero. This zero value means that the known height value will pass unaltered to the vertex of the mesh. But, when the point is between known height values, it will linearly interpolate their heights and then add the output of the noise function on top of the interpolated value to determine the final height of the vertex. A simplified 2D example of this functionality is shown in Figure 7.

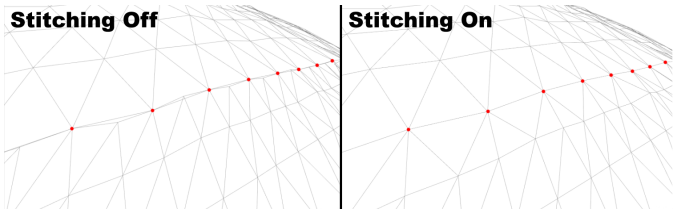

Fig. 8. Surface mesh wireframe screenshots showing the results of mesh seam gap stitching.

5 Mesh Seam Gap Stitching

Like most quadtree systems, the seams between meshes can prove to be problematic. Without any adjustments, the edge triangles can be misaligned, leading to gaps in the meshes. This problem primarily occurs along the border between two meshes of differing levels of detail. To ensure that there are no gaps between any of the meshes of our planets, we designed the system such that when a quadtree mesh splits, the outermost triangles of the four new meshes blend to the previous level of detail. When one of two adjacent same-LOD meshes splits, the edge triangles of the newly split meshes will transition to the level of detail of the mesh before the split, patching the seam between the new meshes and the less detailed meshes around them.

To achieve this result we skip every other vertex along the outer edge of the new mesh. By doing this we create three triangles where there used to be four, allowing a blend from one level of detail to another. This is one of the reasons that the vertex resolution of the meshes is a power of two plus one. If there is a power of two plus one vertex along the edge, then there is exactly a power of two triangles along the edge, allowing it to be easily split and merged by factors of two.

6 Alignment of Normals Along Mesh Seams

Another problem that arises with the quadtree system is aligning the vertex normal vectors between two meshes at different levels of detail or on different faces of the planet. For each vertex generated in a mesh, a normal vector is also generated for graphics purposes (shading based on gradient of normal vectors). If the normal vector for two colocated vertices does not match, there is a discontinuity in the shading of the surface. Because the two meshes are completely independent of each other, the normal vector at a vertex on one mesh may not properly align to the normal vector of the same vertex on an adjacent mesh. This is because the meshes can not use the correct height gradients for normal vector calculations along their edge vertices, since they do not have access to height data outside their mesh.

To ensure that the normal vectors are the same for a vertex on two adjacent meshes, we give each mesh height data for a one-vertex buffer outside that mesh. This allows each mesh to properly calculate the vertex normal vectors along the edge points. This vertex buffer scales with the level of detail, so the sampling width of the buffer matches the sampling width of the rest of the mesh vertices.

7 Materials and Color Calculations

Similar to Digital Elevation Models, the United States Geological Survey also publishes surface mosaics of the celestial bodies in our solar system. These mosaics are cylindrical, true color images of the surface of a planet. Previously, we have used these images as textures stretched across the entire planet. Because these images are typically of a similar resolution as the DEM, they looked good from orbit, but the surface has single color pixels stretching hundreds of meters. Another problem with these mosaics is that they contain shadows, particularly on the north and south poles of the moon. Since the mosaics are a combination of multiple pictures of a planet, the direction of these shadows can be in different directions. Interestingly, there are even sections of the moon that never receive direct sunlight called Permanently Shaded Regions (PSRs). Scientists can only estimate what these areas look like using height data and other readings. Given these constraints, we decided it would be best to design our own materials for the planets, instead of

Fig. 9. Orbital View of the Moon rendered in Unreal Engine 4 using our planetary mesh generation system.

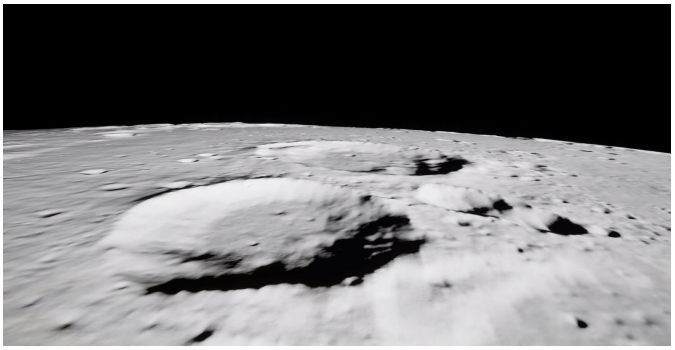

Fig. 10. Surface View of the Moon rendered in Unreal Engine 4 using our planetary mesh generation system.

using USGS mosaic as textures.

The goal is to construct a material that could transition from orbital and surface views while looking realistic from both. Therefore, the material would need to delineate the height map data of the planet to ensure that surface details can be seen from orbit. The material would also need surface variation beyond a simple texture to make it look more realistic from the surface. Finally, the surface texture tiling would have to be limited, because it would become extremely obvious from an orbital view.

To satisfy all of these requirements, the base color of our shader is determined by a few parameters. The first parameter is the height of that fragment on the planet. With this, we can make the peaks of the planet a brighter tone than the recesses to show the depth of surface detail across the planet independently of the level of detail of the meshes. We break down this height value to an 8-bit scale and pass it to the shader via the red channel in the vertex color. To get the base color of the planet, the other parameter we use is the output of a Perlin noise function that represents the ratio of two colors specific to each planet. This produces the effect of the colors blending between each other across the surface of the planet. After studying the surface colors of planets, we determined that interpolating between two colors looks sufficiently realistic since the majority of planet surfaces can be represented within that simplified spectrum. For example, the surface of the Moon and Mercury are mostly a light and a darker shade of gray. The surface of Mars is primarily a light red blended with a dark brown. Even the non ocean surface of the Earth can be simplified to the colors green and a light yellow-orange. With this color ratio, we can add another layer of variation to the planetary surface, and create the base color for fragments across the planet. Similarly to the height value, the ratio of these two colors is passed to the shader through the green channel of the vertex color, while the two colors themselves are passed as parameters directly to the shader.

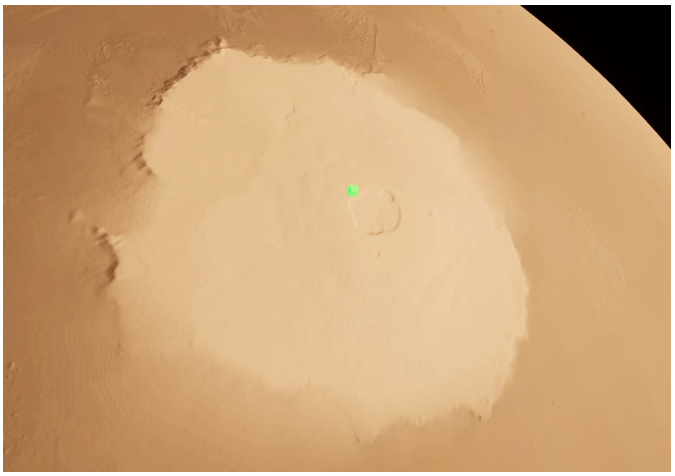

Fig. 11. Mars Rendered with a 10 km cube located at the peak of Olympus Mons, verifying the accuracy of the generated surface.

The final component of our materials rendering is surface textures. We use two image textures and two corresponding normal maps for each planet. One pair is displayed in the recessed areas of the planet and another that is used in the raised areas. These two textures would be blended together using the height value passed in the red channel of the vertex color, similar to the height scaling of the base color. The textures would be distinct, blending from one texture to the other around the threshold altitude. Although the textures are tiled across the surface, if a fragment is too far away from the camera, the texture will not be displayed, only showing the base color. This creates a fading effect for distant texture details and allows the planet to be seen from orbit without any texture tiling, because only the base color will be displayed.

8 Performance

In order to generate surface meshes on the CPU without hitching, where the client update cycle is disrupted by long computation, causing interruption to otherwise smooth rendering, we must use asynchronous function calls. These asynchronous calls, named AsyncTasks in Unreal Engine, transfer parameters to a function called in another thread, which does not directly affect the main thread. At the end of the AsyncTask, we set a flag that mesh generation completed, and the corresponding meshes on the planet are swapped out with our newly generated meshes during the main update loop.

Sampling the data from the DIL and the Perlin noise functions in a separate thread takes about 22 ms on average per mesh of 65x65 vertices on a AMD Rysen 7 1800x Eight Core Processor, 3.90 GHz. This sampling speed is independent of the level of detail of the mesh and primarily determined by the number of meshes being generated at any given time. At the beginning of a simulation, when no mesh has been generated yet, this speed is slightly slower at around 26 ms per mesh. However, later on in the simulation when many of the meshes have already been generated, the mesh data can be sampled within 17 ms.

8.1 Accuracy of Surfaces

The mesh accuracy of our generated planets is very important to the fidelity of our simulations. The main way we have confirmed the accuracy of the mesh on our planets is to pick a known location on

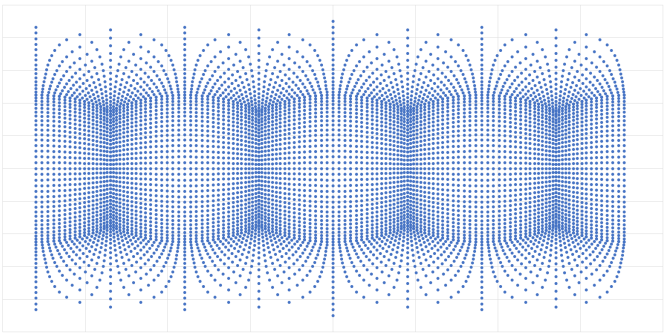

Fig. 12. A graphical representation of the sampling algorithm for turning the original cylindrical DEM into our six DIL files.

a particular planet, such as the highest point of Olympus Mons on Mars, and calculate where that point should be relative to the center of the planet. Olympus Mons is approximately 25 km height and at the coordinates 18.65°N 226.2°E [5]. Given that the radius of Mars is about 3398.5 km, we can use simple spherical coordinates to calculate the position of Olympus Mons. We can then measure that location in our surface simulation using Unreal Engine global coordinates and compare the results. Figure 11 illustrates this confirmation.

8.2 Size of DIL files

Most DEMs are too large to load into memory all at once. For example, the Mars MGS MOLA dataset is 11 GB at a 200 meter pixel resolution around the equator [3]. Because DEMs are stored as cylindrical projections, the resolution of the polar regions are far higher than the equatorial regions, as illustrated by Figure 12. For our application, we need a full dataset at a single resolution. Since the DIL files are a cubic projection of the original data sets, the polar regions of the planet are stored at a similar resolution as the equatorial regions allowing us to thin the excess data at the poles in the original DEMs and store it into a smaller file.

DIL Resolution	2049^2	4097^2	8193^2	16385^2
DILs Storage Size (16-bit data)	50.3 MB	201 MB	805 MB	3.2 GB
Spatial Resolution on Mars (equatorial)	2604.19 m/pixel	1302.41 m/pixel	651.28 m/pixel	325.66 m/pixel

Fig. 13. The table shows how the size of the DIL files and spatial resolution on the planet Mars increase with the increase in resolution of the DIL files.

For comparison, if we reduce the original MGS MOLA dataset to a 325 meter resolution, while maintaining its cylindrical projection, it would occupy 4.17 GB of storage. If we use 16k DIL files, we can store the height data of the entire planet Mars at a 325 meter resolution only using 3.2 GB of harddrive space. This is a vast improvement over storing the full 4.17 GB DEM, while keeping the spatial resolution along the equator constant between the two file formats.

9 Conclusion and Future Work

This paper presented a method for generating planetary meshes in real time for use in full scale simulation environments. Using the Digital Isometric Landscape (DIL) files that we designed, we can generate planetary meshes that accurately represent known celestial bodies down to a variable resolution and render them to scale. Our Perlin noise interpolation algorithm increases the displayed detail of the terrain far beyond that of the original DEM by filling in the gaps of the original height map with procedurally generated terrain. The quadtree level of detail system allows us to generate multiple resolution meshes at runtime, increasing the terrain resolution dynamically where needed. These three components allow a planet to be rendered accurately at multiple scales and is lightweight enough to run in computationally intensive VR engineering simulations.

Moving forward, we have plans to increase the accuracy and improve the aesthetics of our planets. Currently, when the level of detail of the meshes exceeds the resolution of the DIL files, the meshes bilinearly interpolate between the known data points and add Perlin noise. The discrete nature of the bilinear interpolation creates a "checkerboarding" effect with a visible outline of transition areas between two data points. We plan on replacing this with a more realistic bicubic interpolation which will smooth the surface and remove this effect.

As is, our system requires complete surface datasets for celestial bodies we want to model. Unfortunately, many height maps for the celestial bodies in our solar system are incomplete. For example the Venus DEM has large bands of missing data and most of the moons in our solar system only have small patches of available height data, if any at all. Future work will expand this system's capabilities to include realistically filling in the gaps of nearly complete DEMs with procedurally generated terrain at the macroscopic level. Conversely, there are many highly detailed maps for small subsets of the Moon and Mars, such as the Apollo landing sites or the landing site of the Curiosity Mars rover, Gale Crater. We plan on expanding our system to seamlessly integrate these highly detailed datasets when possible, increasing the fidelity of our simulations based in those areas.

Lastly, we are currently looking at many ways to increase the performance of our quadtree level-of-detail system. Specifically, we are looking at job invalidation for meshes that no longer need to be generated. When a mesh needs to display a higher resolution of detail, that mesh is pushed to a queue and eventually splits to the higher detail meshes. If the user moves from one area along the surface to another very quickly, all of the meshes in between those two points will be pushed to this queue. This can waste computing time since some of the meshes along the path may no longer need to be at a high level of detail. Although this problem becomes infrequent when planets are rendered at their full scale, we foresee it becoming more of an issue when we start to generate some of the smaller moons of our solar system.

References

[1] J. Bevins. *Libnoise*, 2007. `http://libnoise.sourceforge.net/`.

[2] R. B. D. d'Oliveira, I. P. do E. Santo, and A. L. A. Jr. Procedural planet generation based on derivate fbm noise. 2018. url-http://www.sbgames.org/sbgames2018/files/papers/ComputacaoShort/188242.pdf.

[3] R. L. Fergason, T. M. Hare, and J. Laura. Hrsc and mola blended digital elevation model at 200m v2, 2018. `http://bit.ly/HRSC_MOLA_Blend_v0`.

[4] Y. Livny, Z. Kogan, and J. El-Sana. Seamless patches for gpu-based terrain rendering. *The Visual Compute*, 25:197–208, 2008. doi: 10.1007/s00371-008-0214-3

[5] NASA. Olympus mons. `https://mars.nasa.gov/gallery/atlas/olympus-mons.html`.

[6] S. O'Neil. A real-time procedural universe, part one: Generating planetary bodies. 2001. `https://www.gamasutra.com/view/feature/131507/a_realtime_procedural_universe_.php`.

[7] K. Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985. doi: 10.1145/325165.325247