Large Terrain Modeling and Visualization for Planets

Steven Myint, Abhinandan Jain, Jonathan Cameron, Christopher Lim
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA
Steven.Myint@jpl.nasa.gov

***Abstract*—Physics-based simulations are actively used in the design, testing, and operations phases of surface and near-surface planetary space missions. One of the challenges in real-time simulations is the ability to handle large multi-resolution terrain data sets within models as well as for visualization. In this paper, we describe special techniques that we have developed for visualization, paging, and data storage for dealing with these large data sets. The visualization technique uses a real-time GPU-based continuous level-of-detail technique that delivers multiple frames a second performance even for planetary scale terrain model sizes.**

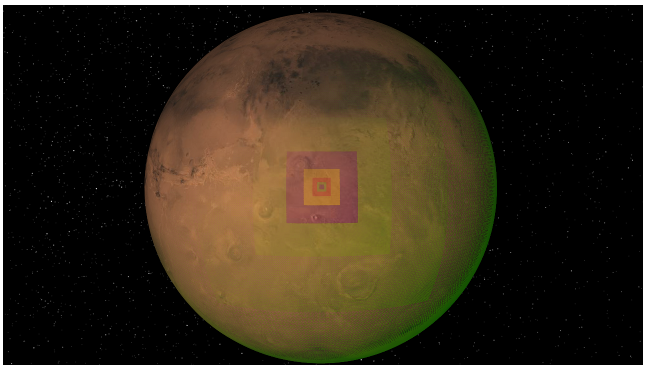


Figure 1: A Mars data set displayed using a continuous level of detail technique. The mesh grid overlaid on the planet, shows the underlying levels of detail.

I. Introduction

NASA's surface and near-surface planetary exploration missions involve landers and surface exploration rovers. These missions require the development and use of closed-loop simulations for the space platforms and the planetary environments. These simulations can involve entry, descent, landing (EDL) [1] and rover traverses over rough terrain scenarios [2]. The simulations are used for, among other things, landing site selection, science planning, and outreach. To model the motion of a rover over a terrain surface, a geometric representation of the surface is essential. We use the SimScape [3] software to handle terrain models of planetary surfaces. We use the Dspace [4] software to visualize the planetary surfaces. While these capabilities provided a highly flexible and capable layer for moderately sized simulation needs, they are inadequate for many planetary-simulations involving large terrain models. For instance, a 10 cm resolution terrain over a 10 km square region is approximately 80 GB in size. Planetary scale data is significantly larger and impractical for the memory/storage resources of most computing platforms.

In this paper, we describe techniques for the real-time modeling and visualization of planetary simulations involving very large terrains. We describe techniques that do not require any special purpose hardware, and are scalable to support data at multiple resolution scales and different extents. We describe how planetary data can be handled seamlessly together with Cartesian digital elevation map (DEM) segments within the same simulation. And we then go on to describe how we efficiently visualize these large terrains using the GPU. We also show how we can overlay additional data onto the terrain visualization.

Section II begins with an overview of the SimScape terrain-modeling library. This is followed by Section III, which describes the SimScape extensions for handling large terrains. Lastly, Section IV explains how we visualize large terrains using a real-time continuous level of detail technique.

II. SimScape Terrain Modeling

The terrain modeling capabilities are built upon the SimScape library, which is briefly described in this section.

A. SimScape Background

With SimScape, we can model surfaces as a DEM, a planet, or an arbitrary mesh. SimScape provides an API to access the data and to transform the data between the various terrain representations. SimScape can import data from standard terrain data formats such as PDS [5], ISIS [6], and GeoTiff [7]. SimScape saves its data into "stores" using the HDF5 format [8]. HDF5 is a file format (along with supporting libraries) that allows large amounts of data to be saved on a file system in a manner so that data slices can be retrieved efficiently independent of the large size of the data sets.

For the applications considered in this paper, there are two types of terrain representations that are relevant: TopoDem (digital elevation map) and TopoPlanet (planetary surface).

1) Digital Elevation Maps (DEMs): When a vehicle explores a relatively small area, it is possible to represent the terrain surface as a DEM, which is a regular rectangular grid of height data. DEM representations are limited to models with only one height value at any row/column grid point in the DEM. This does not allow multiple height values at a single location, which might occur if the terrain includes overhangs. A DEM is sometimes called a 2.5-D surface, meaning that it is not a full 3D surface such as a sphere or ellipsoid (which would require an arbitrary mesh for adequate representation), but is more than a 2-D representation, such as an image. The advantage of using a DEM is that height access is essentially a matrix look-up involving very simple index arithmetic. So accessing height data with a DEM is very fast. On the other hand, looking up height data with a more general mesh is much slower.

Each TopoDem is stored in the HDF5 store along with other configuration data including the physical bounds of the TopoDem, the grid spacing, where the TopoDem is on the surface of the planet, and other related configuration information.

2) Planets: A TopoDem (DEM) is essentially a regular grid of data in a Cartesian setting suitable for representing moderate sized areas. For larger areas, the curvature of the planet becomes a significant issue and Cartesian DEMs cannot be used directly. However, a planetary surface can be modeled with a regular latitude/longitude grid of height data. The TopoPlanet class in SimScape is used for this and uses a grid of height data interpreted in a polar coordinate system. TopoPlanets are also 2.5-D surfaces since any specific latitude/longitude combination can only represent one height.

Both TopoDem and TopoPlanet are derived from a TopoGrid, which handles the representation of gridded data, interpolation, storage, and retrieval. A TopoDem interprets its gridded data as a regular Cartesian grid while a TopoPlanet interprets its gridded data in a polar coordinate system. TopoPlanets have additional methods for dealing with planetocentric and planetodetic planet surface models.

III. Large Terrain Modeling

In working with large terrain data, our goal has been to set up the full terrain data sets (even full planetary scale) in SimScape once, and be able to use them in real-time simulations at any locale on the data set without additional pre-processing. The following sections highlight the key capabilities developed to meet these goals.

A. Paging

The terrain DEM data is saved to an HDF5 file as an array of floating-point values. Since the data can occupy gigabytes of memory, the DEM data is not loaded entirely in memory and a lazy loading algorithm is used to efficiently load only regions of the DEM data.

1) Lazy Loading: When a DEM data point needs to be read, the lazy loader uses the HDF5 built-in "hyperslab" functions to read only a tiny subset ("patch") of the DEM that contains the needed point from the HDF5 file. As an example, a wheeled rover requires only the terrain data in the immediate vicinity of the wheels for use in wheel-soil contact models. Instead of loading the entire DEM, only a patch directly underneath the wheels is loaded. This can be seen in Figure 2, where the highlighted window under the rover is loaded.

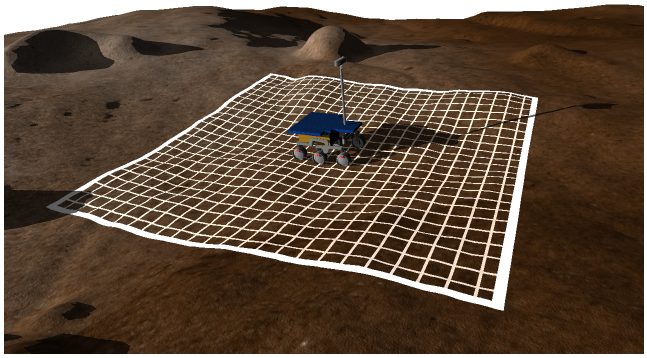


Figure 2: Only a subset of the terrain underneath the rover, highlighted in white here, is loaded from disk when a rover is simulated on the terrain.

2) Caching: As the rover or lander moves, new patches that represent the terrain underneath the vehicle need to be loaded. To increase performance, the patches are saved (cached) in a memory buffer to minimize the need to reload the same patch from the HDF5 file over and over again. The user at run-time specifies the size of the cache. When a DEM point needs to be loaded, the lazy loader first checks if the point exists in the cache. If the point is found in the cache, the lazy loader simply returns the memory address of the patch in the cache that contains the point. If the point is not in the cache, the lazy loader creates a new patch, fills in the patch by reading from the HDF5 file then stores the patch in the cache. The lazy loader keeps track of the number of "hits" for each patch in the cache. To speed up searching for points in the cache, the patches in the cache are sorted by the number of hits. The patches with the most hits are placed toward the top of the cache. If the cache is full, the patch with the least number of hits (i.e. the patch at the bottom of the cache) is deleted first to make room for new patches.

B. Data Tiling

For large data grids, it is inconvenient, and at times impractical to handle the full terrain model data as a single entity. Instead, it is common practice to organize such data sets as a collection of tiles – where each tile is of moderate size. For instance, the MOLA [9] data for all of Mars is

available from PDS servers at multiple resolutions, with the coarsest data consisting of single tiles, while the higher resolution data sets are broken up into multiple tiles. The smaller size of the individual tile data allows them to be more conveniently manipulated and organized. For these reasons, SimScape allows the TopoGrid data to consist of a grid of tiles that span the full data extent. At run-time, the presence of such tiles is transparent to applications. The terrain model API is designed so that access to the underlying data remains the same irrespective of whether the underlying data is organized as a single tile or as multiple tiles. The lazy loading techniques described above are also available with such tiled data sets. However, the added complexity with multiple tiles is that the lazy load page may straddle multiple tiles at any given time.

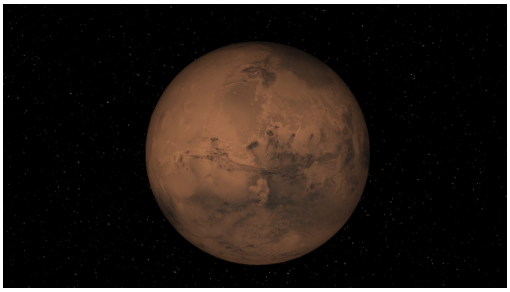
One of the side benefits of using a tiled representation is that the algorithms for creating and transforming the terrain data can often be parallelized to work with individual tiles. Not only does this reduce the memory/disk requirements on the computer, but also can be very important in reducing the time needed to process such large terrain data sets to manageable levels.

IV. Continuous Level of Detail Visualization

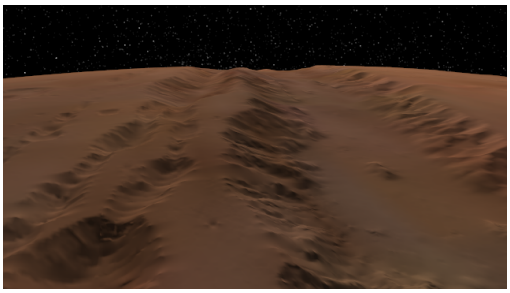
In simulations, we often need to display terrain data sets that are made up of billions of vertices. For example, the MOLA data set, which is displayed in Figure 3, consists of about 1 billion vertices. This amount of data cannot be displayed all at once in a real-time visualization. We need a method of displaying the content that the viewer is interested in, and at the best available resolution. We accomplish this by using a continuous level of detail (LOD) technique called clipmapping [10]. Clipmapping allows us to display high-resolution data where we need it and lower resolution data elsewhere. The placement of the high-resolution area is done dynamically. In conventional LOD techniques, visual artifacts can occur when higher levels of detail pop into the field of view. Clipmapping avoids this and thus results in a smoother rendering. Clipmapping also has a significant performance advantage since most of the computation can be done on the GPU.

Our continuous level of detail (CLOD) system is designed to meet several requirements:

- Be able to display planetary bodies together with high-resolution digital elevation map patches.
- Support the display of large data sets such as ones larger than 100K by 100K, which cannot be loaded fully into graphics memory.
- The terrain data must be displayed in real-time at a frame rate of at least 30 frames per second.
- Allow wheel tracks to be overlaid on the terrains in a dynamic fashion.
- Support displaying terrain data in multiple camera viewports.
- Support the display of multiple data sets together in the same scene.



(a) Mars



(b) We can zoom in to get a detailed view of Valles Marineris

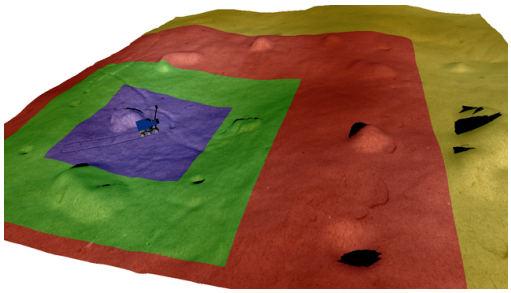
Figure 3: A rendering of the MOLA data set at its highest resolution.

A. Clipmapping

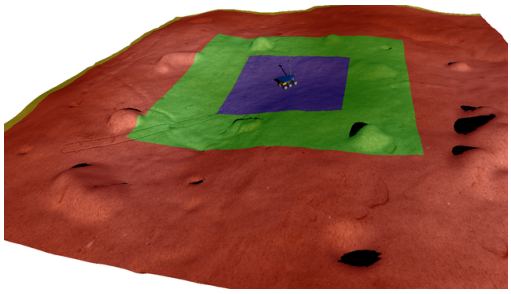
The clipmap approach shows detail at full resolution at a specific location (usually where the viewing camera is pointing) and progressively reduces resolution for areas further away from this location [10]. This is accomplished by organizing the geometry into multiple nested rectangular grids (called clipmap levels), which are centered about some common position. Each successive clipmap level's resolution is reduced by a factor of two as shown in Figure 6.

The clipmapping technique is suitable for implementation on a GPU. This leads to a large performance improvement as GPUs typically have hundreds of cores, which can carry out computations in parallel. The CPU allocates the clipmap levels only once during initialization. (This is more efficient than a more conventional LOD approach, which may require levels of details to be reallocated periodically as the camera view changes.) During render time, as the position of clipmap center is moved around, the GPU does the necessary calculations and adjustments of the vertices of the mesh. The movement of the clipmap center can be seen in Figure 4.

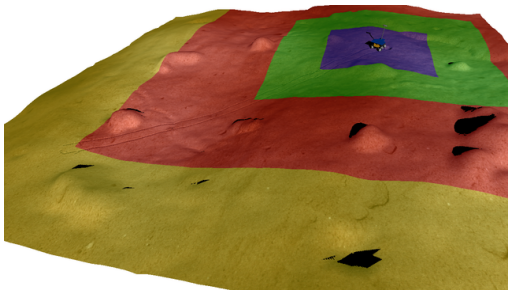
Since the terrain data sets can be very large, they cannot normally be fully loaded into memory. Hence, we rely on terrain lazy loading to load only the required height field data on the fly into memory. Of course, each time a clipmap level is rendered, it will require terrain data of the appropriate



(a) Rover at time t



(b) Rover at time t + 50 seconds



(c) Rover at time t + 100 seconds

Figure 4: The highest-resolution area can be dynamically adjusted. In this case, the high-resolution area is following a rover.

resolution to be available. There are two approaches to getting the coarser levels of terrain data: the first approach is to pre-generate a series of coarser versions of the terrain and register them with the terrain. The appropriate coarser level will be used when rendering a level. The second approach is to work with just the highest resolution terrain data, and to attach lazy loaders at multiple coarseness levels (utilizing HDF5's striding functionality) and load data into memory at the appropriate level of coarseness. While the latter method has the advantage of avoiding the need for pre-processing of the data, it has lower performance since reading HDF5 data at non-native resolutions is considerably slower.

1) Clipmap Levels: Each terrain graphical object is composed of multiple clipmap levels. This is shown in Figure 5. As seen in Figure 6, each clipmap level is composed of a rectangular mesh grid. Each clipmap level has an optional inner grid that can be toggled. The inner grid is disabled for all clipmap levels except for the innermost level.

The innermost level is at the highest resolution of the underlying terrain data. The next outer level is coarser and is at half the resolution. Each clipmap level is constructed as static on the CPU side. To adjust the vertices to conform to the underlying terrain data, height maps are sent in the form of textures to the GPU during render time. The GPU does the necessary calculations to correctly adjust each vertex based on the height map.

For performance reasons, as the viewer gets further from the terrain, we disable the inner clipmaps. We only do this when the viewer is sufficiently far enough such that the change is not noticeable.

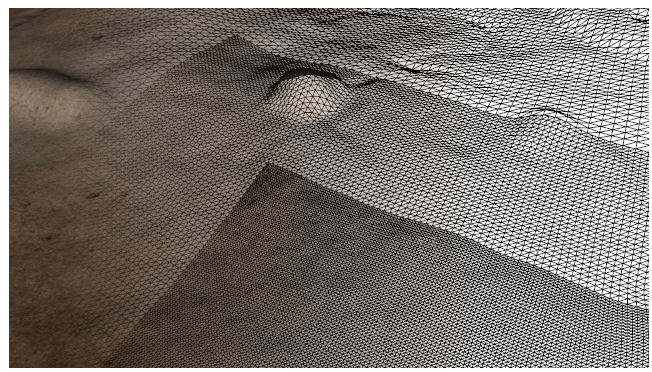


Figure 5: Three clipmap levels displayed in wireframe mode. The level with the finest resolution is closest to the viewer.

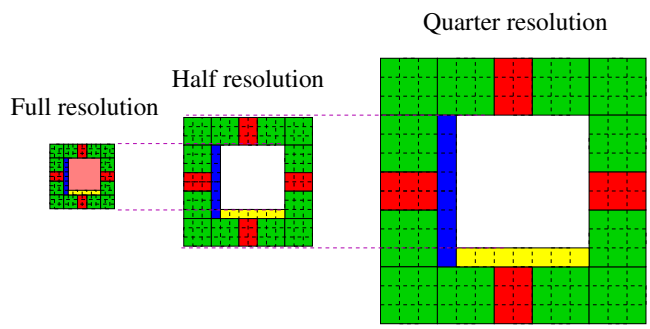


Figure 6: An exploded view of the clipmap levels. The clipmap levels are at progressively higher resolution such that they can be nested within each other.

2) Digital Elevation Maps and Planets: Clipmapping works for both DEMs and planets. As shown in Figure 7, in the case of DEMs, the clipmaps are offset from a plane. In the case of planets, the clipmaps are offset from a spheroid. This is shown in Figure 8.

3) Loading Data: The height map data must be loaded from disk, into memory, and then sent to the GPU.

To avoid loading data every time we move a clipmap, we set the lazy load window to a size larger than the size of the

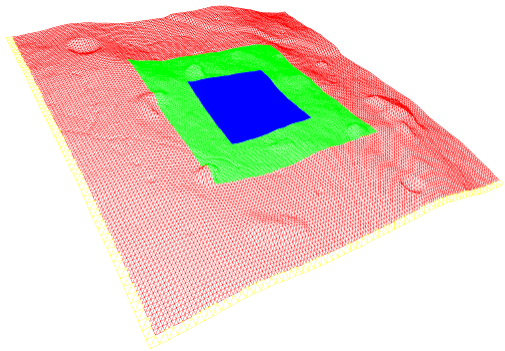


Figure 7: A digital elevation map displayed using clipmaps. The blue is the finest resolution mesh, while the red is the coarsest.

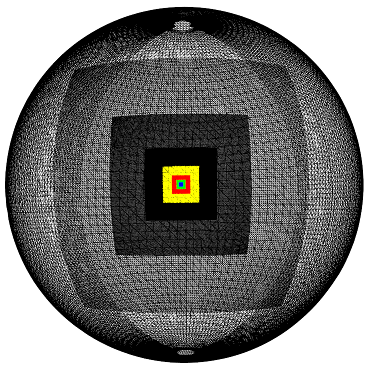


Figure 8: When displaying planets, clipmaps are wrapped around a spheroid.

clipmap. Each time the clipmap moves out of the bounds of the lazy load window, fresh data is loaded from disk.

For very large data sets, the time required to load data from disk can impact rendering performance. This can reduce the visualization responsiveness as the user mouses around to change the camera viewpoint. To avoid this, we load the terrain data in a background thread. This allows the data to be gradually filled in. The order in which the data is loaded is important. The coarsest level clipmap is loaded first. Gradually, the inner clipmaps are filled in. Lastly, the center-most clipmap is loaded.

4) Positioning High Resolution Mesh: During each rendering step, we reposition the clipmaps based on the camera pointing direction and location. This places the high-resolution area at the location the viewer is looking.

Placing the high-resolution area is more involved when multiple cameras are viewing the same terrain. We determine which camera is most active. We then allow that camera to have control of where the highest-resolution clipmap is placed.

Sometimes it is necessary for the highest-resolution clipmap to automatically follow an object of interest; e.g. a rover moving on a rough terrain. Were the rover to be surrounded by coarser regions then it is quite possible that the rover may appear to be hidden under the lower-resolution clipmaps or float above them. For such needs, we allow the highest-resolution clipmap position to be optionally tied to any object in the scene. This includes tying it to the rover as illustrated in Figure 4.

5) GPU Program: Once the data is loaded from disk, almost all of the computation is done on the GPU pipeline including the vertex shader, the geometry shader, and the fragment shader. First, the vertex shader program positions each vertex of each clipmap level. To do this, the vertex shader must take the clipmap's mesh coordinate and convert it to a Cartesian coordinate. To get the x and y position, it first multiplies the mesh coordinate by the resolution of the clipmap grid data. It then offset this value by the current clipmap center position (typically the position that the camera is pointing at). To get the z position, it samples the height map texture. The above procedure is for DEMs, but an analogous procedure is used for planets.

The geometry shader then removes any unneeded vertices. This is only relevant in cases where the terrain has missing data. These areas become holes.

Lastly, the fragment shader computes the color of the pixel based on lighting direction, albedo, and surface normal. The surface normal may be perturbed at this stage to give the impression of a wheel track. Additionally, an optional color overlay can be applied at this stage. Both wheel track and color overlay information is sampled from a texture that can be updated at each render step by the CPU.

6) Seams: Vertical gaps appear at the intersections at which two clipmaps meet. These gaps appear at every other vertex of the finer-resolution clipmap. This can be seen in Figure 9. We generate triangles to fill in the gaps. These triangles act as a skirt that surrounds each clipmap level.

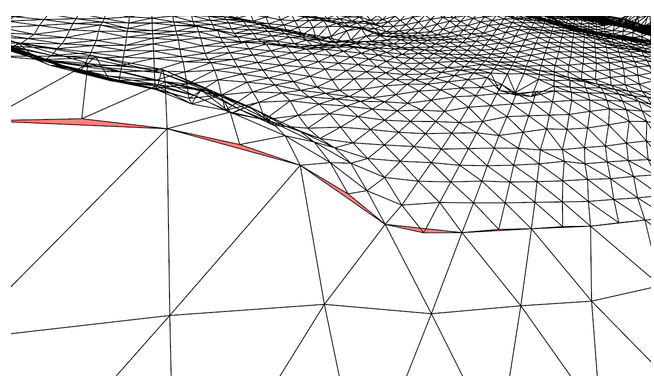


Figure 9: The seams, where the two clipmap levels meet, are highlighted in red.

B. Combining Multiple Data Sets

Planetary terrain data may come from more than one source. To visualize the multiple data sets, we use a scene graph. The scene graph describes the spatial relationship between the terrain data sets in a hierarchical tree. In Figure 10 and Figure 11, we depict a scene that is composed of the Sun, Earth, Mars, and the Martian Victoria Crater detail site. The Sun is at the root of the scene graph. Earth and Mars are children of the Sun with some position and orientation offset. The Victoria Crater model is a child of the Mars planet data with a corresponding offset transform.

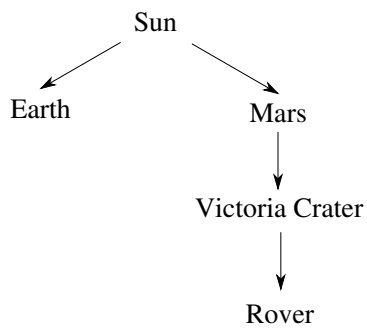
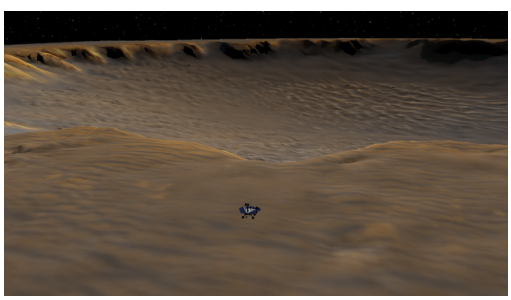
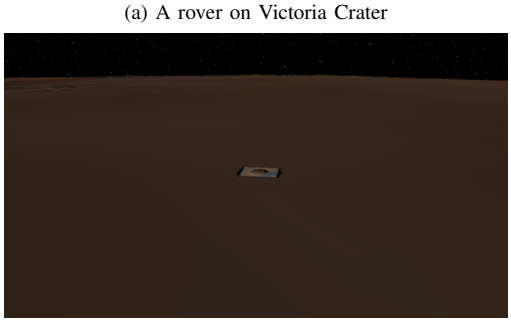


Figure 10: A scene graph that shows connectivity. Each edge in the graph contains information about relative position and orientation.



(a) A rover on Victoria Crater



(b) Zoomed out version of the above image, which shows Victoria Crater in the context of the rest of Mars

Figure 11: An example of positioning multiple terrains using a scene graph.

The example described above is approximately how our simulation arranges the data. But this type of arrangement of the objects can lead to significant floating-point error when visualizing such large scenes. This error manifests as jitter in the position of the camera and graphical objects. In current graphics cards, coordinates are stored as single precision floating-point. This limited amount of precision is not enough to simultaneously handle both the vehicle-scale coordinates and the planetary-scale coordinates. To resolve this problem, the simulation reorganizes the scene graph for the visualization in such a way that the camera can remain close to the origin of the scene. This is shown in in Figure 12. Branches of the scene graph that contain large distances are separated. Depending on which object the user is interested in, the scene is rearranged such that the object of interest is placed at the origin of the scene. All other objects in the scene are translated relative to that object. In this way, the coordinates of the objects near the camera and the camera itself remain small and the jitter problem is resolved.

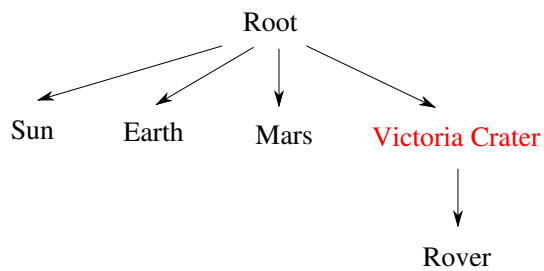


Figure 12: A scene graph that is optimized for visualization. Victoria Crater is placed at the origin of the scene, and the rest of the objects are translated accordingly.

C. Overlays

By default, a texture image representing albedo is applied to the rendered terrain. But, we also support the application of additional overlays for more realistic visuals. One example of this is the support for overlaying wheel tracks onto the terrain. This is utilized for rover simulation scenarios where wheel tracks are important visual cues for the rover's traverse path. An example of this is illustrated in Figure 13. We also have implemented support for applying structured noise to add a visual sense of roughness at the small scale for low-resolution terrain data.

Synthetic overlays can also be used to provide additional information to the user. For example, we have support for applying a color-coded height map to the terrain as shown in Figure 14. We can also draw longitude and latitude lines on the terrain.

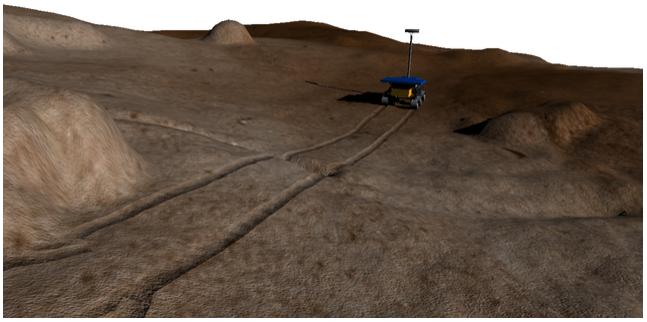


Figure 13: Wheel tracks overlaid onto a terrain dynamically as a rover drives.

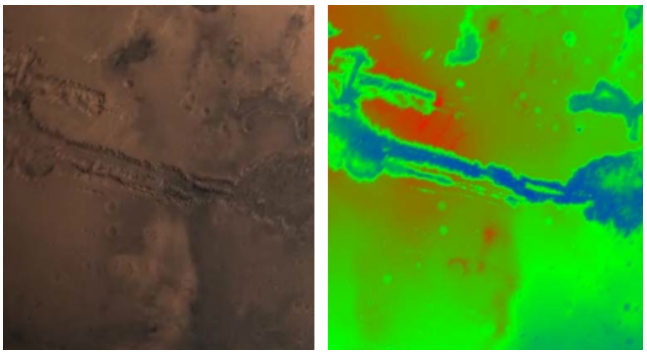


Figure 14: A color-coded height map overlaid onto a terrain.

V. Conclusion

The techniques described in this paper enable the use of large terrain data in real-time simulations for surface planetary missions and overcome the serious size limitation when working with planetary terrain simulations. Inhomogeneous terrain data at multiple resolutions and with different representations can be used together. We can handle DEMs and planets together in the same simulation. And we can visualize all this in real time by making use of the GPU's large number of parallel cores. Additionally, we can add various overlays onto the terrains like wheel tracks and color maps. All this is done without the use of any special computing hardware.

In the future, we plan to add support for clipmapping of not just geometry, but also textures. This will allow us to support displaying very large textures instead of being limited to the GPU's maximum texture size, which is typically 8K by 8K pixels. These textures will be used for albedo maps, color overlays, and normal maps.

Acknowledgment

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration

References

[1] J. Balaram, R. Austin, P. Banerjee, T. Bentley, D. Henriquez, B. Martin, E. McMahon, and G. Sohl, "DSENDS-A high-fidelity dynamics and spacecraft simulator for entry, descent and surface landing," in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 7. IEEE, 2002, p. 7.

[2] H. Nayar, B. Balaram, J. Cameron, M. DiCicco, T. Howard, A. Jain, Y. Kuwata, C. Lim, R. Mukherjee, S. Myint, A. Palkovic, M. Pomerantz, and S. Wall, "Surface Operations Analyses for Lunar Missions," in *AIAA Space*, August 2010.

[3] A. Jain, J. Cameron, C. Lim, and J. Guineau, "SimScape terrain modeling toolkit," in *Space Mission Challenges for Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on*. IEEE, 2006, p. 8.

[4] M. Pomerantz, A. Jain, and S. Myint, "Dspace: Real-Time 3D Visualization System for Spacecraft Dynamics Simulation," in *Space Mission Challenges for Information Technology, 2009. SMC-IT 2009. Third IEEE International Conference on*. IEEE, 2009, pp. 237–245.

[5] "PDS: The Planetary Data System." NASA, 2011. [Online]. Available: http://pds.nasa.gov

[6] "USGS Isis: Planetary Image Processing Software." USGS, 2011. [Online]. Available: http://isis.astrogeology.usgs.gov

[7] "GeoTiff." OSGeo, 2011. [Online]. Available: http://trac.osgeo.org/geotiff

[8] "HDF5." HDF Group, 2011. [Online]. Available: http://www.hdfgroup.org/HDF5

[9] "Mars Global Surveyor: MOLA." NASA, 2011. [Online]. Available: http://pds-geosciences.wustl.edu/missions/mgs/mola.html

[10] F. Losasso and H. Hoppe, "Geometry clipmaps: terrain rendering using nested regular grids," in *ACM SIGGRAPH 2004 Papers*. ACM, 2004, pp. 769–776.