SimScape Terrain Modeling Toolkit

Abhinandan Jain, Jonathan Cameron, Christopher Lim, John Guineau
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109

Abstract— **Planetary space mission applications involving landers and surface exploration vehicles make extensive use of terrain models within their simulation testbeds. Such terrain models are large, complex and have a variety of attributes including topography, reflectivity, soil mechanics, and hazard information. Sources for the terrain models include planetary data archives, field tests, and analytically constructed models. Simulation uses of such models include surface rover vehicles' kinematics and dynamics models, instrument models, camera models and robotic arm models.**

This paper describes the SimScape middleware toolkit for providing a common infrastructure to represent terrain model data from multiple data sources and make them available to simulation applications. SimScape simplifies the overall simulation design by eliminating the traditional need for custom terrain model interfaces to terrain data sources for simulation users. SimScape provides a collection of libraries and tools to use and manage terrain environment models within the simulation applications.

I. Introduction

Many types of planetary exploration missions require simulations that involve software models of the terrain surface of planetary bodies. Missions that involve rovers operating on the planet's surface require terrain models with rover simulations to develop flight software and test mission operations plans. Space missions that land a payload on a planet's surface also require terrain models for realistic radar models for developing on-board landing control software.

Surface rover simulations require rich, complex terrain models involving a wide range of surface attributes including topography, surface imagery, soil mechanics properties, hazard conditions, and optical characteristics such as reflectivity. First, the terrain model must represent basic surface topography to compute the location of the ground under each rover wheel. At the same time, the soil mechanics properties are needed to compute the forces acting on the rover wheels for wheel slippage and sinkage models. These computations are carried out for each wheel several times for each simulation step. Fast algorithms that work with the terrain models are needed for such demanding applications. Planetary exploration rovers use imagery from on-board cameras to plan motions. Creating simulated images for on-board cameras also requires terrain models with appropriate surface optical properties such as imagery and reflectance.

Landing a mission on a planetary surface is a risky and difficult process. Extensive planning and simulation is necessary to develop landing control software and to reduce the risks during landing. Terrain models needed for landing simulations involve basic topography as well as other types of specialized terrain information. For instance, laser altimeters may need topography and surface reflectivity. Radar models also need specialized properties such as ground density to estimate the strength of radar returns. Recent landing missions often include landing approach control based on on-board camera imagery. Construction of suitable images leads to requirements for surface optical properties similar to those for rover camera operations. Science teams also need high-quality terrain models for simulating science instruments and planning surface operations.

These simulation applications need to incorporate raw terrain data from a variety of sources including planetary data archives, field tests, synthetic models (derived from existing terrain models) as well as algorithmically generated ones. Such primary terrain data needs to be processed and assembled into a terrain model prior to use by applications. This often requires piecing together data from different sources, processing relative spatial transforms among them, assigning attributes to the geometry, handling different data formats and representations and using height field and/or 3D models as appropriate

The **SimScape** terrain modeling toolkit described here arose out of the recognition that while the different terrain applications represent a wide diversity in terrain modeling and algorithm requirements, there is also a significant overlap in the requirements among them. Our goal was to create a common toolkit that could be used and shared by a wide range of simulation applications. A high level description of the requirements for **SimScape** are:

- Support multiple representations of the terrain geometry. These include 2.5D digital elevation map grid representations, point cloud representations, 3D mesh representations, and 2.5D triangulated irregular network (TIN) mesh representations and support for geo-referenced planetary data models.
- Support algorithms and methods for transformations between different terrain model representations.
- Support use within applications as a general-purpose, embeddable library library with efficient algorithms and methods for operating on terrain models.
- Support terrain model data from different sources including planetary archives, field test sites, and synthetic and analytic terrain models (Figure 1).
- Support exporting and importing terrain models to and from a variety of standard terrain data formats including PDS, GeoTiff, USGS ISIS and VRML.

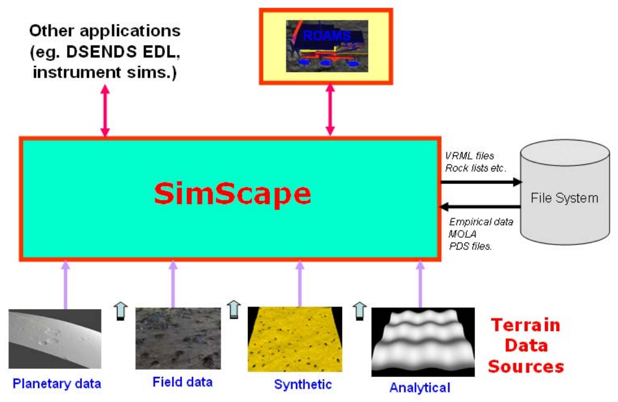
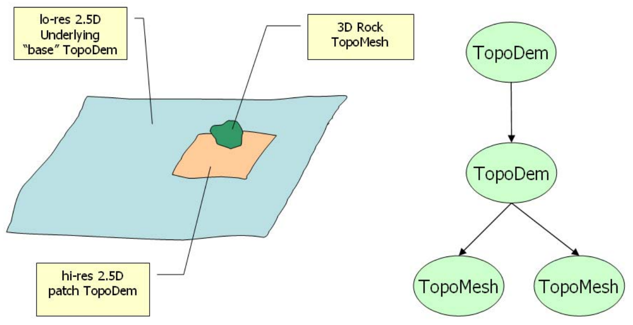


Fig. 1. **SIMSCAPE** as middleware for terrain models within simulation applications

- Support the use of overlays of surface properties such as material composition, texture, albedo, terra-mechanics parameters, reflectivity, and user defined properties onto the underlying terrain geometry.
- Support for composite terrain models assembled from heterogeneous component terrain models, eg., base terrains together with component 3D rock models (Figure 2). This includes the ability to assemble terrain models from a variety of topographic components in a tree hierarchy with relative transforms to rotate, position and scale the subcomponents.



lo-res 2.5D Underlying "base" TopoDem
3D Rock TopoMesh
hi-res 2.5D patch TopoDem
TopoDem
TopoDem
TopoMesh
TopoMesh

Fig. 2. Assembling a terrain site model

- Support persistent storage and retrieval of assembled terrain models for later retrieval and use by applications.
- Provide an open architecture to allow extension of the library with new algorithms and terrain model types by users.

There are many products that provide some of these required capabilities. But none cover all of the needs of terrain modeling for planetary space missions. Daylon Graphics offers a stand-alone product called Leveller [2] that provides an ability to construct and edit 3D terrains. The GNU Triangulated Surface Library (GTS) [1] has many useful functions for modeling surfaces with sets of triangular faces, but does not deal with gridded elevation data. The Geospatial Data Abstraction Library (GDAL) [4] supports importing of raster terrain data from a wide range of data formats. The Planetary Data System (PDS) [5] provides planetary images and data collected by various NASA missions and related software. The Integrated Software from Images and Spectrometers (ISIS) [6] provides software for processing data fro orbiters and instruments. There are also many Geographic Information Systems products available, such as GRASS [7], that provide useful tools for geographic data analysis.

This paper describes the **SIMSCAPE** middleware toolkit that has been developed to meet the simulation requirements described above. **SIMSCAPE** provides a common infrastructure for importing terrain model data from multiple data sources and making them available to simulation applications. These terrain data sources include planetary and empirical terrain data sets, terrain synthesis models and analytical models. The **SIMSCAPE** infrastructure simplifies the overall simulation design by eliminating the traditional need for custom terrain model interfaces to terrain data sources and simulation users. **SIMSCAPE** provides a collection of libraries and tools to use and manage terrain models within spacecraft simulation applications. As described in more detail below, **SIMSCAPE** uses some of available open-source tools and libraries described above to develop an integrated terrain modeling capability for use by simulation applications.

Section II provides an overview of the **SIMSCAPE** design architecture, object types and their functionality. Section III describes algorithms and methods for manipulating the **SIMSCAPE** objects as well as for transforming them into one another. Section IV describes the various data formats supported for importing and exporting terrain data. Section V describes support for storing and retrieving terrain models to and from the file system. Section VI describes user interfaces for visualizing terrain models and Section VII describes simulation applications using **SIMSCAPE**.

II. SIMSCAPE OBJECT HIERARCHY

As shown in Figure 3, **SIMSCAPE** provides a class hierarchy that supports various terrain modeling needs. Broadly, these classes fall into the following categories:

- Classes to represent the topographic geometry of terrain.
- Classes to overlay surface properties onto the terrain geometry.
- Container classes to assemble component terrain models into hierarchical models.
- Classes to store and retrieve terrain models from the file system.

Before examining classes in each of these categories, we will briefly discuss the base class from which they are derived.

All storable **SIMSCAPE** classes are derived from the **COREOBJECT** class. Each class derived from **COREOBJECT** implements methods to save and restore their object-specific data from the persistent store. For more details on persistent stores see Section V.

The base class **COREOBJECT** also provides functions to assign and retrieve meta-data about the object, referred to by

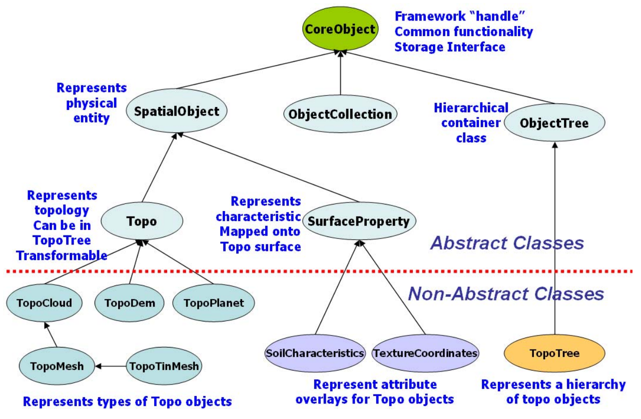

CoreObject
Framework "handle"
Common functionality
Storage Interface
Represents physical entity
SpatialObject
ObjectCollection
Hierarchical container class
ObjectTree
Represents topology Can be in TopoTree Transformable
Topo
Represents characteristic Mapped onto Topo surface
SurfaceProperty
Abstract Classes
Non-Abstract Classes
TopoCloud
TopoDem
TopoPlanet
TopoMesh
TopoTinMesh
SoilCharacteristics
TextureCoordinates
TopoTree
Represents types of Topo objects
Represent attribute overlays for Topo objects
Represents a hierarchy of topo objects

Fig. 3. The SimScape class hierarchy

SimScape as "attributes". There are two types of attributes. The first type of attributes are user-defined attributes. Their purpose is to provide a way for users to store and retrieve textual data about an object such as labels, annotations, data sources, etc. CoreObject provides the capability to save and retrieve arbitrary text strings with keys. A second type of attribute provides simple access to known data internal to each topographic object. For instance, the number of samples for a TopoDem can be accessed via a read-only attribute with the key 'samples'. Other known attributes such as the textual description of the TopoDem (that has the key 'description') can be accessed or modified via the attribute get/set functions. These two types of attributes use the same underlying mechanisms to store their information in the persistent store so that objects retrieved from the persistent store will include all of their attribute data.

We begin by examining the primary topographic object classes.

A. Primary topographic object classes

1) TopoDem: One of the primary topographic classes is the TopoDem. As the name implies, this is primarily a DEM (Digital Elevation Map). The elevation values are laid out in a regular x-y rectangular grid. For each x-y value pair, there is only one elevation (Z) value. The TopoDem provides data access and interpolation functions to compute the elevation value for a specified x-y location. Each TopoDem contains information about the extent (physical area) that the data grid covers.

DEM topographic data contain grids of regularly spaced elevation data. In some cases, each elevation data point represents an average elevation in the grid square (or rectangle) that contains the point. This type of data is called `CENTERPIXEL`. In other cases, the elevation data denotes elevation at one of the corners of the grid square. This type of data is called `FENCEPOST`. TopoDem supports both types of elevation data representations.

TopoDems can be created from elevation data stored in files (or persistent stores, explained in more detail in Section V). The second source of data for TopoDems is via an analytic construction capability. TopoDems can be created with a number of ideal geometric surfaces such as flat surfaces, sloped surfaces, wavy surfaces (sine/cosine wave function), and other similar shapes. Users can also define new analytic surfaces.

TopoDems also provide many functions that provide useful data beyond simple elevation lookup. These include functions to compute the normal to the surface at a specified location, compute statistics for a small patch surrounding a specified location, and other functions geared towards ground vehicle simulations.

Although the most common use of the TopoDem class is to provide access to a regular, Cartesian grid of elevation data, the coordinate types can be more general. For instance, TopoDem coordinates can represent radius and angle coordinates for a polar grid. Similarly, TopoDem coordinates can also be latitude and longitude pairs in the TopoPlanet class (see Section II-A.5 for more details about TopoPlanets). TopoDem is based on a C++ template and can contain elevation data in `double`, `float`, or even integer types.

2) TopoCloud: A TopoCloud is simply a collection of x-y-z points called vertices. TopoClouds support adding new vertices, merging TopoClouds, and other related functions. A cloud of points may be the 3D range data obtained from stereoscopic image correlation.

3) TopoMesh: A key limitation of the TopoDem is that it is 2.5 dimensional (meaning that there is only one elevation data value for each x-y location). For some vehicle simulations, a truly 3D surface class is necessary to model terrain surfaces with realistic 3D characteristics such as overhangs. The TopoMesh class provides full 3-dimensional irregular mesh capability. The surface is broken up into a set of "Faces" (which are usually triangles) and the vertices necessary to define each face. The TopoMesh class derives from the TopoCloud class (which provides basic vertex storage and operations). The TopoMesh type provides functions to access/add/delete faces, merge TopoMeshs, and refine (subdivide) existing faces to reduce the size of all faces below a desired threshold.

4) TopoTinMesh: For 2.5D terrain data, the TopoMesh class provides the advantage of being economical in the number of polygons needed to represent the surface, while the TopoDem class provides fast look up of elevation values. The TopoTinMesh class combines these advantages for 2.5D terrain models. A TopoTinMesh is an irregular mesh with faces and vertices that is constrained to be 2.5D by the vertex/face construction process. This means that a TopoTinMesh provides the unambiguous elevation lookup of a TopoDem with the data flexibility of a TopoMesh. A well-constructed TopoTinMesh represents the same elevation data as a TopoDem with considerably reduced fewer polygons and reduced storage requirements with minor run-time elevation lookup penalties.

5) TopoPlanet: The TopoPlanet class is the primary end-user class for dealing with the surface of a planet.

A **TopoPlanet** represents the surface of a planet in a generalized way by adding an offset to an nominal planetary radius value based on some idealized reference surface model (such as a sphere or an ellipsoid). To accomplish this, **TopoPlanet** uses the **PlanetModel** class to compute radius values from a latitude and longitude pair or convert from latitude and longitude pairs to x,y,z Cartesian locations (and vice versa).

The reference surface accounts for the fact that planets are not in perfectly spherical in general, but are usually elliptical or oblate in shape, with flattening at the poles relative to the equator. Using a fixed reference surface also allows the software to make use of the limited precision of floating point number formats for describing the surface detail by factoring out a large constant reference value associated with planetary radii.

PlanetModel is a generalized base class. It contains an object of the **PlanetSurfaceReference** class that is the base class for planetary reference surfaces and knows how to compute a reference surface radius for any specified (planetocentric) latitude and longitude pair. **SimScape** provides two simple derived classes, **Ellipsoid** and **Sphere**, that model a spherical and ellipsoidal reference surface, respectively.

There are two classes derived from **PlanetModel**: **PlanetDetic** and **PlanetCentric**. These two classes do the primary work of adding offsets to the reference surface radius. **PlanetCentric** assumes the offset is added to the reference surface in a direction radially outward from the center of the planet. **PlanetDetic** assumes the offset is added in a direction normal to the local reference surface at the specified location. The **PlanetModel** class also uses the class **PlanetCoordInfo** (and several related classes) to take care of whether longitudes increase in the eastward or westward directions, and how to deal with longitude wrapping.

To provide the offset elevation data at any specified latitude and longitude, the **TopoPlanet** class uses an embedded **TopoDem** object. For this usage, the typical meaning of x and y of the **TopoDem** are replaced by longitude and latitude (respectively) and the elevation values of the **TopoDem** are interpreted as offsets from the reference surface.

When a **TopoPlanet** object is created, the user must first create an appropriate **PlanetModel** object (using the **PlanetDetic** class or a **PlanetCentric** class) with the appropriate reference surface (such as **Ellipsoid**). The resulting **PlanetModel** object is used to create the **TopoPlanet** object and is installed as a data member of the **TopoPlanet** object for future surface location computations.

The **TopoPlanet** class provides various useful functions such as the ability to compute the radius or elevation offset of the surface at a specified latitude and longitude, compute local surface normals, and compute the intersection of a ray with the surface.

The design of the **TopoPlanet** class and the classes that it uses provides a flexible and extensible way to implement complex planetary topographic systems.

B. Container Types - Trees and Nodes

In order to support the assembly of topographic models from various sources the **SimScape** framework provides a **TopoTree** tree class populated with **TopoTreeNode** objects. The topology of the tree structure is a simple downward branching tree. Each **TopoTreeNode** object can contain a set of **TopoTreeNode** children. Each **TopoTreeNode** contains a single topographic data object such as a **TopoDem** or a **TopoMesh**. The relative position and rotation of the node (with respect to its parent node) can be specified using an affine transform (which contains a position offset, scale, and a rotation matrix). The position/rotation offset applies to the node, all its children, and to the contained topographic object. This provides the ability to position all topographic objects as needed.

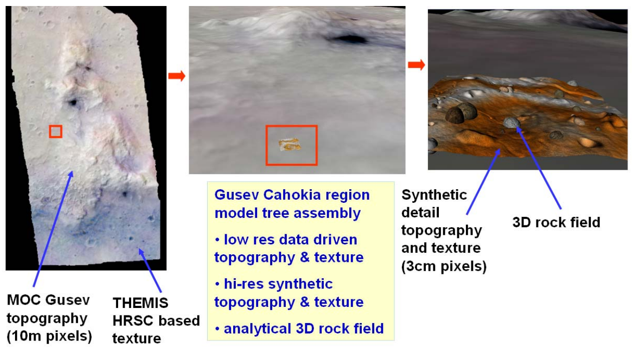

MOC Gusev topography (10m pixels)
THEMIS HRSC based texture
Gusev Cahokia region model tree assembly
• low res data driven topography & texture
• hi-res synthetic topography & texture
• analytical 3D rock field
Synthetic detail topography and texture (3cm pixels)
3D rock field

Fig. 4. Mars Gusev site terrain mesh with detail DEM and 3D rock meshes

C. Surface Property Overlays

Elevation data is essential to vehicle simulations but is not the only type of data needed to describe an area of terrain. Important science and engineering data related to terrains are available and must be associated with terrains so that the data can be accessed for locations within the terrain. Examples include images to be overlaid on the surface for visualization purposes, soil properties at specific locations (eg. soil types, cohesion), and spectral properties of the terrain surface. These 2D raster data sets are called surface property "overlays" in **SimScape** and are derived from the **SurfaceProperty** base class. Multiple surface property overlays may be associated with a **Topo** object.

SimScape allows overlaid surface properties to be specified in frames different from that of the parent **Topo** object. In order to access a surface property value at a **Topo** vertex, the overlay data set must be "bound" to the vertices on the parent **Topo** object. The "binding" process (see Figure 5) registers the overlay (including a position/rotation offset) with the parent topographic object and initializes various internal data structures for accessing the overlay data. Each overlay provides its own accessor functions that return **SurfaceProperty** values for specific locations (x-y or x-y-z depending on the type of topographic object). For instance, an image to be overlaid over a topographic object is called

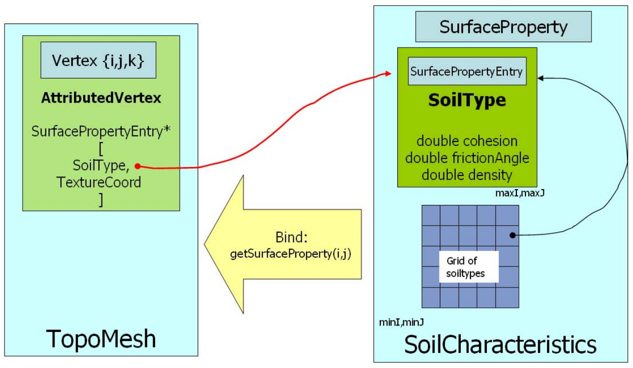


Fig. 5. Binding of surface attributes to the TOPO surface objects

a **TEXTUREOVERLAY**. The **TEXTUREOVERLAY** class is derived from **SURFACEPROPERTY** and provides a primitive data type called **TEXTURECOORDINATE** (containing an s,t image coordinate pair). When the bound **TEXTUREOVERLAY** information is needed, it can be accessed from the parent topographic object and **TEXTURECOORDINATE**s can be generated for specified x-y coordinates.

In the process of binding surface properties on to topographic objects, a coordinate map can be specified. If the topographic object is essentially 2.5 dimensional (for **TOPODEM**s, for instance), a simple one-to-one correspondence between the surface property set and the underlying topographic object might be appropriate. However, if the topographic object is 3 dimensional (like a **TOPOMESH**), controlling how to map the surface property to the 3D object may not be as simple. Several coordinate mapping classes are provided for this purpose (and the user can create new mapping classes derived from one of the existing ones). For instance, the class **COORDAFFINEMAP2D** can introduce an arbitrary affine transform (offset, rotation, and skew) in the mapping process. See the left part of Figure 6 for an example including skew. The user can also define a custom mapping using the **COORDCUSTOMMAP2D** class as shown in the right part of Figure 6.

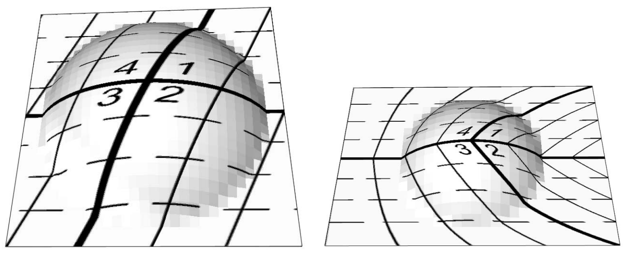


Fig. 6. 2D Coordinate Mappings For Binding of Surface Properties

Interesting and useful mappings for 3-dimensional figures are also available, including a 2D mapping, a spherical mapping using **COORDSPHERICALMAP3D** (see left image in Figure 7) and a user defined custom mapping using **COORDCUSTOMMAP3D** (see the right image in Figure 7 and note that the same texture is mapped to each face of the cube). Although these example surface property mappings are shown for textures (images), the same capability can be used to map other types of surface properties onto topographic objects.

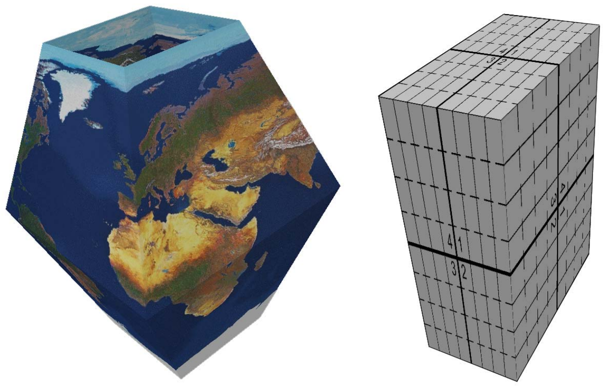


Fig. 7. Left: 3D Coordinate Mappings For Binding of Surface Properties

It is also possible to bind an overlay (along with relative position/rotation offsets) to **TOPOTREE** nodes. Such overlays are applied to the **TOPOTREENODE** sub-tree.

D. Architecture Extensibility

SIMSCAPE supports extensibility by the mechanism of runtime dynamic loading of extension libraries. The user can extend any of the base classes provided by **SIMSCAPE** and the new classes can be easily loaded at run time so that the new functionality is available without recompiling the base **SIMSCAPE** library. Each new class also provides basic functions to save its data to a persistent store so that its classes can be saved and restored from the persistent store like any of the basic classes of **SIMSCAPE**. The new user-defined classes can be extensions of existing classes or be created to perform a single function such import data from external formats ("Importers"), export data to external formats ("Exporters"), convert an object from one type to another ("Transformers"), or modify an object without changing its type ("Manipulators"). See Figure 2. More details about these types of objects will be covered in Section III and following.

III. TRANSFORMERS AND MANIPULATORS

Many common operations on basic topographic objects involve some type of manipulation of the data in the topographic object and constructon of a new topographic object of the same type. These functions can be thought of as **manipulators** since the object returned is of the same type as the original, but the internals have been changed or manipulated. (In most cases, the original object has not been modified.) For instance:

- The function **TOPODEM**::getPatch() allows the caller to specify a region or patch of a terrain **TOPODEM** and return the patch as a **TOPODEM**.
- In **TOPOMESH**::refine(resolution), the **TOPOMESH** adds vertices to the mesh as necessary to ensure that no

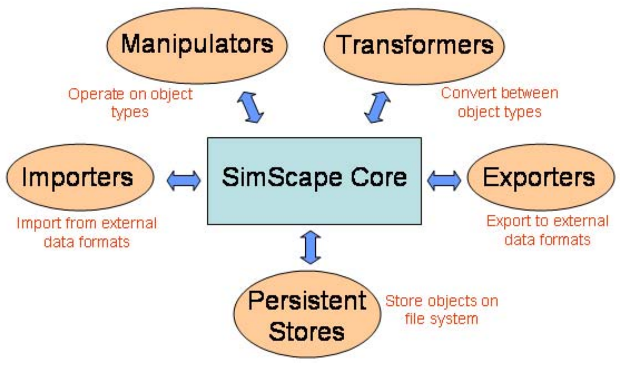

Manipulators
Operate on object types
Transformers
Convert between object types
Importers
Import from external data formats
SimScape Core
Exporters
Export to external data formats
Persistent Stores
Store objects on file system

Fig. 8. Run-time loadable dynamic extensions

vertex is farther away from another vertex than the specified resolution. The **TOPOMESH** is updated in place.

- In **TOPOTREE**::normalize(), the **TOPOTREE** is traversed and a new **TOPOTREE** is constructed for which all transforms are applied to the internal data of the topographic objects (such as vertex data) so that all the transformations become identity transformations and the original transforms are no longer necessary. The normalized tree is more efficient at run-time because the transforms are not needed.

In addition, there is a need at times to transform a terrain model from one type into another. For instance, this can happen if the source data is of a specific type (eg. an irregular mesh) while the application has a need for a 2.5D grid terrain data type. At times, in more complex simulation scenarios, it is possible to use multiple different representations of the same terrain by different modules within the simulation. For instance a **TOPOTREE** representation with component 3D mesh models is very suitable for graphics visualization, while a 2.5D **TOPODEM** version of the same tree may be needed for a wheel/soil dynamics model. Algorithms and methods that convert one model type into another are referred to as **transformers**. Examples of such transformers within **SIMSCAPE** include:

- A transformer that converts a 3D **TOPOMESH** into a 2.5D **TOPODEM** model that takes as arguments a projection plane and a grid resolution for the desired **TOPODEM**. The algorithm projects the irregular **TOPOMESH** onto the plane and samples along the desired grid to create the **TOPODEM**. This transformer makes use of the transformer from a **TOPOMESH** to a **TOPOTINMESH** during the projection process.
- A transformer to convert a normalized **TOPOTREE** into a **TOPODEM**. This transformer essentially merges all the component **TOPO** objects within the **TOPOTREE** into a **TOPOMESH** and then converts it into a **TOPODEM**.

We plan to add additional transformers such as those generating surface property overlays for properties such as roughness, slopes, height fields from a **TOPO** object.

IV. EXPORTERS AND IMPORTERS

The import and export of terrain data for different formats is supported by specialized importer/exporter classes in **SIMSCAPE**. This mechanism allows the addition of support for new formats in the future. Most common data formats are for data in regular grid format that are suitable for creation of **TOPODEM** and **TOPOPLANET** terrain models. Several providers also support the specification of geo-referencing data along with projection information for the data. These data providers also often provide libraries that support processing such data files. **SIMSCAPE** makes use of these libraries where available to allow importing of such terrain models. The key data import formats supported by **SIMSCAPE** include:

- **SIMSCAPE** includes a **Geospatial Data Abstraction Library (GDAL)** [4] based terrain model data importer for the creation of **TOPODEM** terrain models from a variety of data formats. GDAL is an open source software library for raster geospatial data formats. As a library, it presents a single abstract data model to the calling application for all supported formats. The GDAL library has been used to develop an importer extension to allow importing of a large number of widely used data formats [8] into **SIMSCAPE**. This importer can also handle raster data from several image format files.
- The **Planetary Data System (PDS)** [5] is a publicly available archive that distributes scientific data from NASA planetary missions, astronomical observations, and laboratory measurements. The data is available in the PDS format with header information describing the data sets. The PDS site also provides software for processing the PDS data and has been used to develop a **TOPOPLANET** importer for **SIMSCAPE**.
- The **Integrated Software for Imagers and Spectrometers (ISIS)** [6] provides a tool for processing, analyzing, and displaying remotely sensed image data. ISIS primarily handles 2-D image data (as single-band cubes) and 3-D data (as multi-spectral or hyper-spectral cubes) from imagers and spectrometers. **SIMSCAPE** imports ISIS data by using the ISIS toolkit to export data into TIFF format followed by the use of the GDAL importer to complete the import.
- **SIMSCAPE** also can import simple raster format data consisting of ASCII and binary height field data into a **TOPODEM** object.
- **SIMSCAPE** also can import range map data generated from stereo camera pairs into a **TOPOCLOUD**. Such data is typically obtained by camera sensors on surface exploration rovers.

Exporting terrain models into external data files is supported in **SIMSCAPE** as follows:

- **SIMSCAPE** uses the GDAL library for exporting **TOPODEM** model data into external data formats. The export formats in the GDAL library supports are a subset of those supported for importing. All of these formats are available to **SIMSCAPE** via the GDAL library.

- To facilitate 3D graphics visualization, **SimScape** supports the export of all **Topo** object data into VRML files.

V. Persistent Store

A persistent storage system allows **SimScape** objects (surfaces, rocks, textures, etc.) to be saved to a storage device for later retrieval. The persistent storage class ("Store") in **SimScape** can be thought of as an "input/output stream"; objects can be read and written to the stream and the **PersistentStore** class takes care of loading or storing the object's binary data from/to storage.

The **PersistentStore** class has an abstract device interface to support different types of storage. Currently **SimScape** uses an XML schema to save objects, but other storage types such as SQL databases and even flat files can be implemented in the future. The resulting **PersistentStore** storage files are portable and can be transported between different machines that support **SimScape**.

The **PersistentStore** is extensible and can support any C++ class type derived from **CoreObject**; new **SimScape** objects can be added without the need for recompiling the entire **SimScape** library. This is implemented by requiring each **SimScape** object to register methods to load and save itself to the **PersistentStore**. The list of loading and saving methods is keyed by the object's class type. To load an object from the store, the **PersistentStore** searches the list of registered classes for the specified class type and invokes the necessary load methods to reconstruct the object. Since a **SimScape** object may be derived (subclassed) from another **SimScape** object, each **SimScape** object also saves its complete class hierarchy (type chain) to the store. When restoring an object from the store, the most specialized class available is used to create the object. Each **SimScape** object gives itself a unique ID string to distinguish itself from other **SimScape** objects of the same type.

Methods for saving and reloading objects can be loaded and registered at run-time using the dynamic loading capability described earlier. Therefore users can create new classes that can be saved to persistent stores.

When writing to the store, each **SimScape** object also saves a version number; when loading from the store the **SimScape** object can therefore detect whether the stored object was created with a newer or older version of **SimScape** and process it appropriately.

As mentioned earlier, the **PersistentStore** currently stores objects into an XML file. The XML schema is implemented as follows:

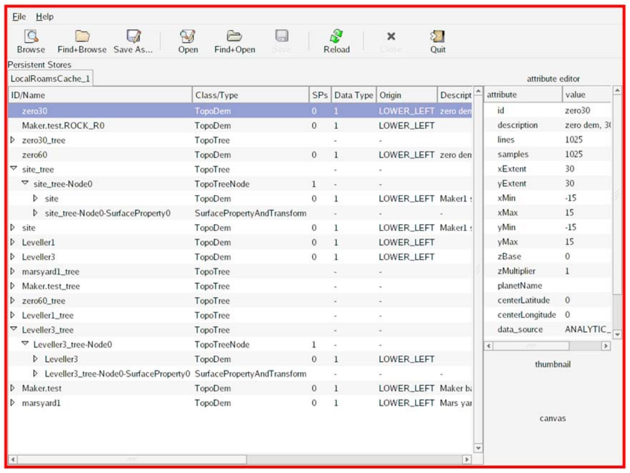
1) Objects are saved in a tree structure in the XML file with each child object saved as a leaf (node) of its parent object's node so reconstructing an object with child objects is simply a matter of walking the tree.
2) Arrays of simple data types (integers, strings and floating point numbers) are compressed and stored in separate files to save space. The path to the array file is stored in the XML file.
3) Large data files (eg., image files) are also stored in separate files with the paths kept in the XML file.

VI. User Interface

The **SimScape** toolkit includes a GUI for browsing and editing terrain models as well as a 3D visualization interface based on a JPL-developed 3D graphics toolkit called Dspace for visualizing the terrain models.

While **SimScape** provides a portable and well-defined C++ interface, a Python [9] scripting interface is also available for all the object classes. The Python interface is auto-generated using the **SWIG** [10] wrapper generator tool. The **SimScape** Python classes closely mimic the underlying C++ classes. **SimScape**'s Python interface is very useful for creating scripts to implement functions needed to prepare and manipulate **SimScape** terrain models. Such scripts have been used to develop a regression test suite as well as a host of tutorial examples for using the **SimScape** objects.

The Python scripting interface has also been used to develop a GUI browser for **SimScape** persistent stores (Figure 9). The browser allows a user to browse and edit the contents of



ID/Name	Class/Type	SPs	Data Type	Origin	Descript
zero30	TopoDem	0	1	LOWER_LEFT	zero den
Maker.test.ROCK_R0	TopoDem	0	1	LOWER_LEFT	
zero30_tree	TopoTree		-	-	
zero60	TopoDem	0	1	LOWER_LEFT	zero den
site_tree	TopoTree		-	-	
site_tree-Node0	TopoTreeNode	1	-	-	
site	TopoDem	0	1	LOWER_LEFT	Maker1 :
site_tree-Node0-SurfaceProperty0	SurfacePropertyAndTransform		-	-	-
site	TopoDem	0	1	LOWER_LEFT	Maker1 :
Leveller1	TopoDem	0	1	LOWER_LEFT	
Leveller3	TopoDem	0	1	LOWER_LEFT	
marsyard1_tree	TopoTree		-	-	
Maker.test_tree	TopoTree		-	-	
zero60_tree	TopoTree		-	-	
Leveller1_tree	TopoTree		-	-	
Leveller3_tree	TopoTree		-	-	
Leveller3_tree-Node0	TopoTreeNode	1	-	-	
Leveller3	TopoDem	0	1	LOWER_LEFT	
Leveller3_tree-Node0-SurfaceProperty0	SurfacePropertyAndTransform		-	-	-
Maker.test	TopoDem	0	1	LOWER_LEFT	Maker ba
marsyard1	TopoDem	0	1	LOWER_LEFT	Mars yar

attribute	value
id	zero30
description	zero dem, 3(
lines	1025
samples	1025
xExtent	30
yExtent	30
xMin	-15
xMax	15
yMin	-15
yMax	15
zBase	0
zMultiplier	1
planetName	
centerLatitude	0
centerLongitude	0
data_source	ANALYTIC_

Fig. 9. The **SimScape** GUI browser

one or more **SimScape** stores. The GUI also displays the defined attributes for the objects. Several of the manipulator and transformer methods available for the objects can be invoked from the GUI as well. The GUI also provides hooks to visualize a 3D graphics model of any of the **Topo** objects.

VII. Applications

We briefly describe a few simulation applications that currently use **SimScape** for their terrain modeling needs. **SimScape** is in use by the ROAMS rover simulator [11], the DSENDS entry, descent and landing simulator [12] and instrument simulators. Beyond the terrain model definition, **SimScape** provides high-performance algorithms for embedded use by these simulations.

- **ROAMS** (Rover Analysis, Modeling and Simulation) [11] is a physics-based simulation tool for the analysis, design, development, test and operation of rovers for planetary surface exploration missions. ROAMS provides a modular rover simulation framework to facilitate use by planetary exploration missions for system engineering studies, technology development, and mission operation teams. ROAMS currently is being developed and used by NASA's Mars Program as a virtual testing ground for various rover subsystems and components. ROAMS is capable of modeling vehicle dynamics, engineering sensors and actuators, and operational environments. The terrain environment and surface property overlays needed for the wheel/soil dynamics models, the onboard camera sensors, and the 3D graphics visualization are provided by **SIMSCAPE**.

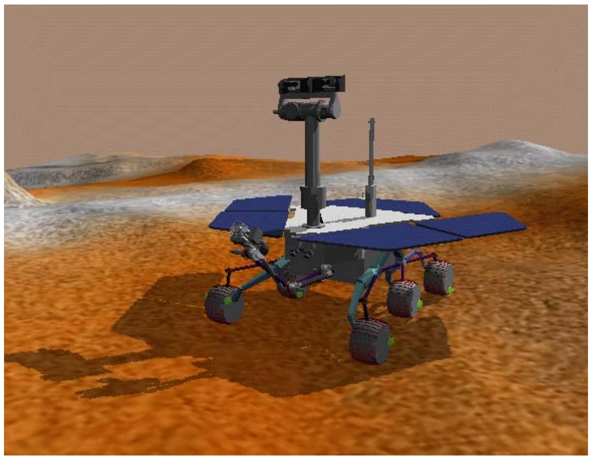


Fig. 10. Roams - Rover Vehicle Simulation Environment

- **DSENDS** [12] is a high-fidelity spacecraft simulator for Entry, Descent and Landing (EDL) on planetary bodies. DSENDS (Dynamics Simulator for Entry, Descent and Surface landing). DSENDS is currently in use by the JPL Mars Science Laboratory project to provide a high-fidelity testbed for the test of precision landing and hazard avoidance functions for future Mars missions. **SIMSCAPE** provides the terrain modeling capability for DSENDS' radar altimeter and landing hazard sensing models.

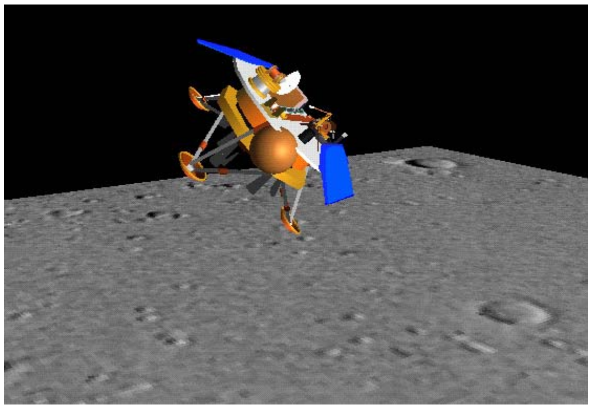


Fig. 11. DSENDS - Mars Entry Descent and Landing Simulation

VIII. CONCLUSIONS

We have described the design and use of the **SIMSCAPE** terrain modeling toolkit that has been developed to serve as a general-purpose framework for meeting the terrain modeling needs of various simulation applications. **SIMSCAPE** is already in use by several key simulation applications for space missions such as planetary landing and surface operations. **SIMSCAPE**'s ability to handle a variety of different terrain model types, import/export data formats, and support for assembling hierarchical terrain models is proving to be very useful in addressing the diverse set of terrain modeling needs that arises within sophisticated physics-based simulations. The **SIMSCAPE** implementation has adopted open-source toolkits where available in its design. **SIMSCAPE** provides both a portable C++ implementation, as well as a Python binding to facilitate its use from within scripts. Future **SIMSCAPE** developments will continue to mature the existing classes and algorithms, as well as develop various extensions as needed.

ACKNOWLEDGMENTS

We would like to thank Meemong Lee and James Wood for their many suggestions during the design and implementation of the **SIMSCAPE** software and Michael Wagner for the implementation of the **SIMSCAPE** GUI. The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration, and has been supported partly by the National Science Foundation Grant ASC 92 19368.

REFERENCES

[1] "GNU Triangulated Surface Library (GTS) website." http://gts.sourceforge.net/.
[2] "The Daylon Leveller Heighfield/Bumpmap/Terrain Modeler website." http://www.daylongraphics.com/products/leveller/.
[3] "Terrain Server, Client, and Maker website." http://terrain.jpl.nasa.gov/.
[4] "GDAL - Geospatial Data Abstraction Library website." http://www.remotesensing.org/gdal/.
[5] "The Planetary Data System (PDS) website." http://pds.jpl.nasa.gov/.
[6] "Integrated Software for Imagers and Spectrometers (ISIS) website." http://isis.astrogeology.usgs.gov/Isis2/isis-bin/isis.cgi/.
[7] "Geographic Resources Analysis Support System (GRASS) website." http://grass.itc.it/.
[8] "GDAL Raster Formats." http://www.remotesensing.org/gdal/formats_list.html.
[9] "Python website." http://www.python.org/.
[10] "Simplified Wrapper and Interface Generator (SWIG) website." http://www.swig.org/.
[11] A. Jain, J. Guineau, C. Lim, W. Lincoln, M. Pomerantz, G. Sohl, and R. Steele, "Roams: Planetary surface rover simulation environment," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2003)*, (Nara, Japan), May 2003.
[12] 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 *IEEE 2002 Aerospace Conf.*, (Big Sky, Montana), Mar. 2002.