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—**Space mission applications involving landers and surface exploration make extensive use of terrain models within their simulation testbeds. Such terrain models are large, complex and involve a variety of attributes including topography, radiosity, soil mechanics, and hazard properties. Sources for the terrain models include planetary data archives, field tests, mathematically constructed models. Simulation users of such models include surface rover vehicles' kinematics and dynamics models, instrument models, camera models, robotic arm models etc. While each of these types of terrain models has certain unique aspects, there is also a large degree of commonality among them.**

This paper describes the SIMSCAPE middleware toolkit that has been developed recently to provide a common infrastructure for importing terrain model data from multiple data sources and making them available to simulation applications. 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 environment models within a wide range of simulation applications.

I. INTRODUCTION

Space mission applications involving planetary landers and surface exploration make extensive use of terrain models within their simulation testbeds. Simulations of surface exploration rovers use terrain models as part of wheel slippage/sinkage dynamics models, for camera and sensor simulation models, and for collision checks. Lander spacecraft simulations use terrain models for altimeter models, radar models, and hazard detection. Science related studies use terrain models for instrument specific simulations.

Such terrain models are large, complex and involve a variety of attributes including topography, radiosity, soil mechanics, hazards properties and other domain-specific information. Sources for the terrain models include planetary data archives, field tests, synthetic models (derived from existing terrain models) as well as analytic ones (created mathematically). Simulation users of such models include surface rover vehicles' kinematics and dynamics models, instrument models, camera models, robotic arm models etc. While each of these uses of terrain models has certain unique aspects, there is also a large degree of commonality among them.

This paper describes the SIMSCAPE middleware toolkit that has been developed recently to provide 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 environment models within spacecraft simulation applications.

There are many products that offer parts of what we need in SIMSCAPE but none that combine all the features that we need. For instance, Daylon Graphics offers a product called Leveller [2] that provides an ability to construct or edit 3D terrains. It does not provide a run-time library to use the resulting terrains in realistic applications. The GNU Triangulated Surface Library (GTS) [1] has many useful functions for modeling surfaces with triangles, but does not have capabilities to deal efficiently with gridded elevation data. GTS also has no tools for importing terrain data from a variety of geographic sources. The Geospatial Data Abstraction Library (GDAL) [4] has many tools for importing from a wide range of geographic data sources. GDAL primarily uses raster bands (sets of gridded data) to represent the wide range of data in can assimilate. It does not deal with irregular meshes and has no provision for hierarchical topographic structures, as SIMSCAPE does. Similarly, the Planetary Data System (PDS) [5] provides planetary images and data collected by various NASA missions and some related software. The Integrated Software from Images and Spectrometers (ISIS) [6] provides software for modeling images and related instruments. Both PDS and ISIS have unique and useful contributions, but neither addresses many of the needs of vehicle simulation. Several of these products provide useful capabilities that SIMSCAPE incorporates directly via their software libraries. See Section IV for more details. There are also many Geographic Information Systems products available, such as GRASS [7], that provide useful tools for geographic data analysis, but do not address many of the needs of vehicle simulation.

Goals of SIMSCAPE are:

- Support run-time use of terrain models in simulations
- 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. In addition, SIMSCAPE includes support for geo-referenced planetary data models. The SIMSCAPE library provides several methods to transform terrain models between these representations.
- Support for composite terrain models assembled from heterogeneous component terrain models, e.g. base terrains with component 3D rocks (Figure 1). This includes the ability to assemble terrain models from a variety of to-

pographic components in a tree hierarchy with offsets and rotations to position the subcomponents appropriately.

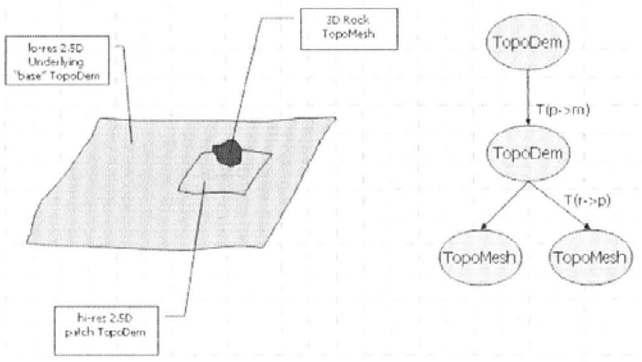


Fig. 1. Assembling a terrain site model

- Support the use of overlays of surface properties such as material composition, texture, albedo, terra-mechanics parameters etc. **SIMSCAPE** supports the overlay of attributes such as reflectivity, terra-mechanics, material etc. onto the underlying terrain geometry.
- Support the use of terrain model data from many different sources including planetary, empirical, synthetic and analytical terrain models (Figure 2). The terrain data can include archival planetary data, field test sites (eg. Mars Yard), synthetic and analytical terrain data.

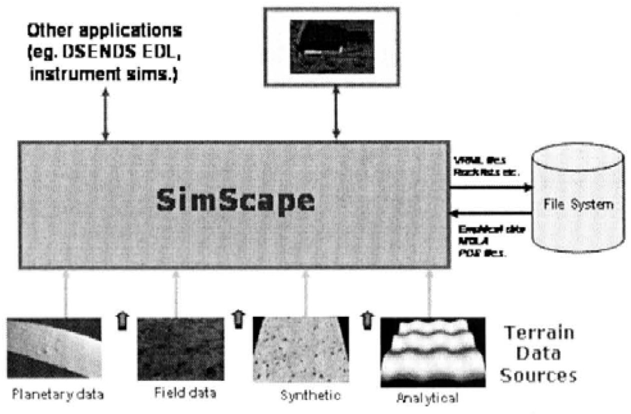


Fig. 2. **SIMSCAPE** as middleware for terrain models within simulation applications

- Support algorithms and methods for transformations between the different terrain model types
- Support the exporting and importing of terrain models to and from the variety of standard terrain data formats, inluding PDS, GeoTiff, USGS ISIS, VRML etc.
- Provide a run-time library with efficient algorithms and methods for use of the terrain models within applications.
- Provide clear interfaces for use as a general-purpose, embeddable library by different applications.
- Provide an extensible architecture to allow extension of the **SIMSCAPE** library with new algorithms and terrain model types by users.
- Support off-line preparation and creation of complex terrain models
- Support persistent storage and retrieval of terrain models for run-time access.

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 VI describes user interfaces for visualizing terrain models and Section VII describes simulation applications using **SIMSCAPE**.

II. ARCHITECTURE AND DESIGN

Since every application area has special needs for terrain data that cannot be guessed in advance by the designers of the **SIMSCAPE** architecture, it is important that the design capture common basic terrain needs, as well as provide the extensibility necessary for end users to extend the framework to meet their specific needs.

A. Extensibility by Plugins

One of the key ways that **SIMSCAPE** provides extensibility is via the mechanism of "plugins". 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. The user adds the compiled object files for their newly derived classes to a plugin directory and instructs the **SIMSCAPE** framework to search the plugin directory to discover the new classes. Each new class would also provide 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 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 ("Exporter"), convert an object from one type to another ("Transformers"), or modify an object without changing its type ("Manipulator"). See Figure 1. More details about these types of plugins will be covered in Section III and following.

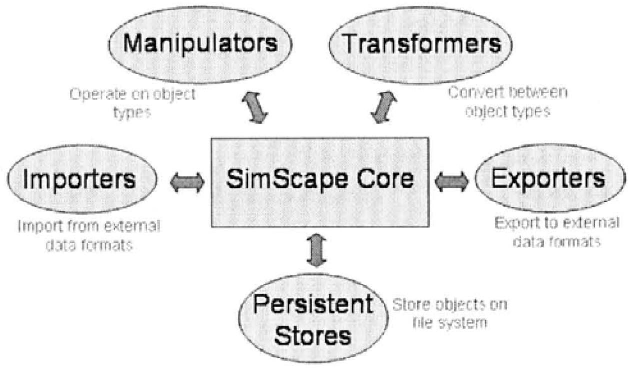

Manipulators
Transformers
Operate on object types
Convert between object types
Importers
SimScape Core
Exporters
Import from external data formats
Export to external data formats
Persistent Stores
Store objects on file system

Fig. 3. Plugins based extensions

B. **SIMSCAPE** *Architecture*

One of the main design goals for the **SIMSCAPE** framework is to provide a tree-like structure to contain the terrain-related objects necessary to model complex surfaces for various types of simulations. With this in mind, there are two types of classes of objects in **SIMSCAPE**: (1) basic topographic primitives (such as DEMs) and (2) container objects (related to the tree structure). Note that all of these classes are derived from a single base class called "CoreObject" to enable storage and retrieval with persistent stores. See Figure 4 for an overview of the class hierarchy. We will start by examining the primary topographic object classes.

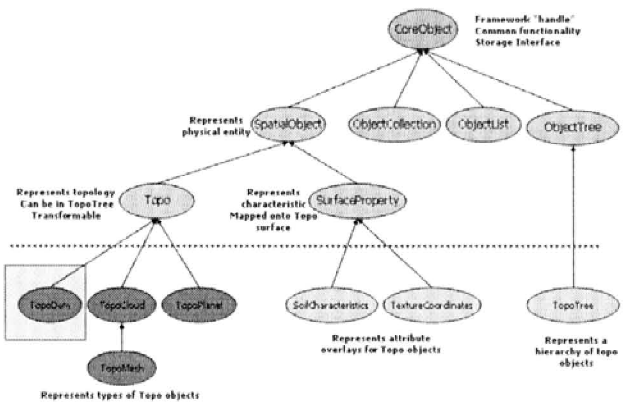


Fig. 4. The **SIMSCAPE** class hierarchy

C. *Primitive 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.

There are currently two sources of elevation data that may be used to create **TOPODEM**s. The primary one is stored elevation data. The persistent store saves the grid of elevation data values along with all necessary information to reconstruct the coverage of the **TOPODEM**. The second source of data for **TOPODEM**s is via an analytic construction capability. **TOPODEM**s 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.

Many sources of topographic data provide grids of regularly spaced elevation data. Exactly how these elvation data values should be interpreted varies. In some cases, each elevation point represents an average elevation in the grid square (or rectangle) the contains the point. In **SIMSCAPE**, this type of data is called `CENTERPIXEL`. In some cases, the elevation data is to be interpreted as the elevation at one of the corners of the grid square. In **SIMSCAPE**, this type of data is called `FENCEPOST`. **TOPODEM**s support both types of elevation data representations.

TopoDems also provide a series of 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 dominant use of the **TOPODEM** type is to provide access to a regular, rectangular grid of elevation data, the class itself is more general. A **TOPODEM** provides a container for any 2-D, regularly spaced data. For instance, the TopoDem could contain elevation data for radius and angle data on a polar grid. Similarly, TopoDems are used to contain the elevation data for latitude and longitude pairs in the TopoPlanet class (see Section II-C.5 for more details about **TOPOPLANET**s). Since the data contained by a **TOPODEM** is based on a C++ template, it can be applied various numerical classes such as `double`, `float`, or even integer types.

2) TopoCloud: A **TOPOCLOUD** is simply a group (or "cloud") of x-y-z points called vertices. **TOPOCLOUD**s support adding new vertices, merging **TOPOCLOUD**s, and other related functions. A cloud of points may be the primary type of data for a variety of applications. An example is the 3D ranging data 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 many 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 similar to 3D meshes used to model surfaces in popular graphics libraries. 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 **TOPOMESH**s, and refine (subdivide) existing faces to reduce the size of all faces below a desired threshold.

4) TopoTinMesh: One of the key operations needed from a terrain models by a ground vehicle is to determine the elevation for a specific x-y location. This is easy to do with a **TOPODEM** since it is inherently a 2.5D structure, but problematic with a **TOPOMESH** since it is 3D and could produce two (or more) elevation values for a single x-y location. The **TOPOTINMESH** class implements a "Triangular Irregular Network" based on Delaunay triangulation. 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 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. 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.

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 shape 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).

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.

The reference surface accounts for the fact that planets in general are not perfectly spherical, but are usually elliptical or oblate in shape, with flattening at the poles relative to the equator. Using a fixed reference surface also allows 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.

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 that radially outward 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 design of the **TopoPlanet** class and that classes that it uses provides a flexible and extendable way to implement more complex planetary topographic systems.

6) Container Types - Trees and Nodes: In order to provide maximum flexibility to assemble topographic data from various sources the **SimScape** framework provides a tree structure populated by **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**.

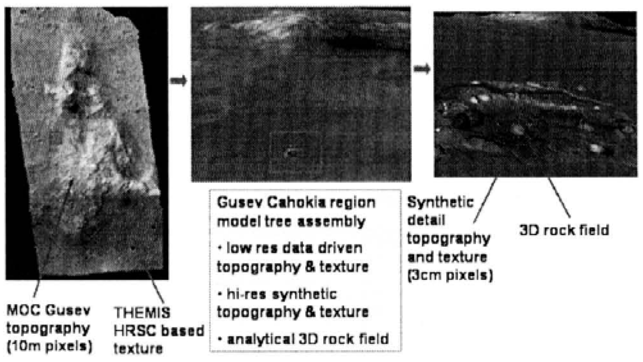

Gusev Cahokia region model tree assembly
• low res data driven topography & texture
• hi-res synthetic topography & texture
• analytical 3D rock field
Synthetic detail topography and texture (3cm pixels)
3D rock field
MOC Gusev topography (10m pixels)
THEMIS HRSC based texture

Fig. 5. Mars Gusev site terrain mesh with detail DEM and 3D rock meshes

The relative position and rotation of the node (with respect to its parent node) can be specified using a "Transform" object (which contains a position offset and a rotation matrix). The position/rotation offset applies the node, all its children, and the contained topographic object. This provides complete flexibility to position all topographic objects within the tree as needed.

An object of the **TopoTree** represents the entire tree structure and contains the tree of **TopoTreeNodes**. It also provides several functions for manipulating the entire tree (such as normalization). Conceptually, the **TopoTree** and **TopoTreeNode** classes could be merged, although they are distinct in our current implementation.

D. Surface Property Overlays

Elevation data is essential to vehicle simulations but is not the only type of data useful to describe an area of terrain. Important science and engineering data related to terrains are often available and must be associated with terrains so the data can be accessed for locations within the terrain. Examples include images to be overlaid on the surface for visualization purposes, the nature of the soil at specific locations (soil types, cohesion, etc), 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 such surface property overlays may be associated with a **Topo** object.

SimScape allows such overlay properties to be specified in frames different from that of the parent **Topo** object. In order however 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 6) registers the overlay (including a position/rotation offset) with the parent topographic object and sets up 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

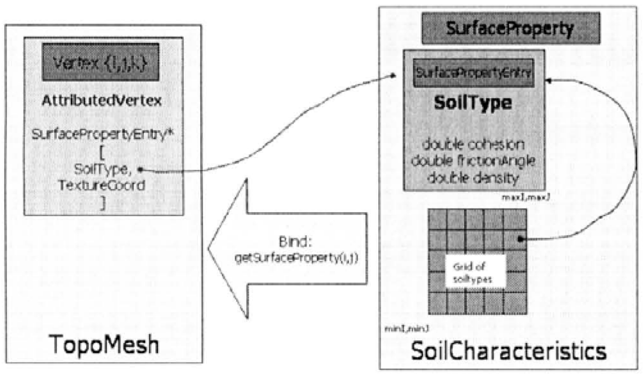


Fig. 6. Binding of surface attributes to the **TOPO** surface objects

depending on the type of topographic object). For instance, an image to be overlaid over a topographic object is called 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 TopoDems, 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 is not so 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 7 for an example. The user can also define a custom mapping using **COORDCUSTOMMAP2D** as shown in the right part of Figure 7.

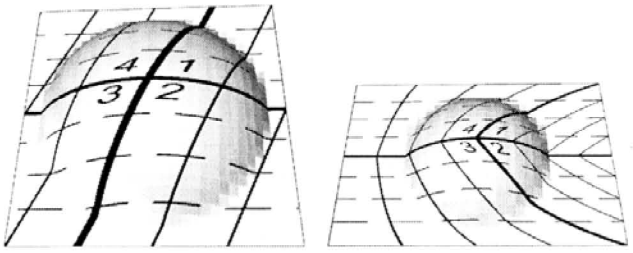


Fig. 7. 2D Coordinate Mappings For Binding of Surface Properties

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 8) and a user defined custom mapping using **COORDCUSTOMMAP3D** (see the right image in Figure 8 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.

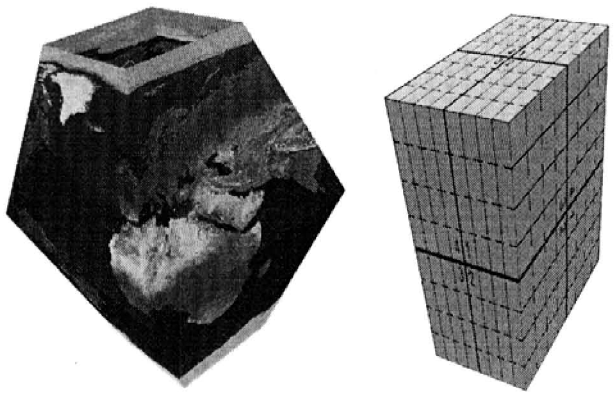


Fig. 8. Left: 3D Coordinate Mappings For Binding of Surface Properties

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.

E. Attributes

The base class CoreObject not only provides the basic functionality to save objects to persistent stores, it provides a set of functions relating to meta-data about the object, referred to here as "attributes". There are two types of attributes with two different purposes. One 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 type of 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.

III. TRANSFORMERS AND MANIPULATORS

Many common operations on the basic topographic object involve performing some type of manipulation of the data in the topographic object and returning 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/patch of a terrain **TOPODEM** and return the patch as a **TOPODEM**.

- In **TopoMesh**::refine(resolution), the **TopoMesh** adds vertices as necessary to ensure that no 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 tranform a terrain model from one type into another. This can happen for instance 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 simulations. 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. This transformer 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

SimScape has been designed to build terrain models from data available in various different formats. The import and export of data for the different formats is supported by plugin importer/exporter extensions for **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 data formats for creation of **TopoDem** and **TopoPlanet** terrain models. Several providers also support the specification of geo-referencing data along with projection information to interpret this data. These data providers also often provide libraries that support the processing of such data files. **SimScape** makes use of these libraries where available to allow the importing of such terrain models.

The key data import formats supported by **SimScape** include:

- The **Geospatial Data Abstraction Library (GDAL)** [4] 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 import extension to allow the import of a large number of widely used data formats [8] into **SimScape**. **SimScape** includes a GDAL based terrain model data importer for the creation of **TopoDem** terrain models from a variety of data formats. 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 multispectral or hyperspectral 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 a simple raster format consisting of ascii height field data into a **TopoDem** object.
- **SimScape** also can import range map data generated from a stereo camera pair into a **TopoCloud**. Such data typically is obtained by camera sensors on surface exploration rovers.

Exporting of 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 data formats that the GDAL library supports exporting is a much smaller 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

SimScape provides a local cache for managing and sharing terrain data products. SimScape provides APIs for handling a whole host of terrain attributes including topography, texture, feature lists etc. properties. **SimScape** also includes classes to efficiently interact with and manipulate the terrain model information at run-time.

A persistent storage system allows **SimScape** objects (surfaces, rocks, textures, etc.) to be saved to a storage device for later retrieval even after the original objects are deleted.

The persistent storage class ("Store") in **SIMSCAPE** can be thought of as an "input/output stream"; objects can read and write to the stream and the PersistentStore class will take care of loading or storing the object's binary data from/to storage.

PersistentStore 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; 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 with 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 objects for the 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 it's complete class hierarchy (type chain) to the store. Each **SIMSCAPE** object gives itself a unique ID string to distinguish itself from other **SIMSCAPE** objects of the same type.

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 act appropriately.

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

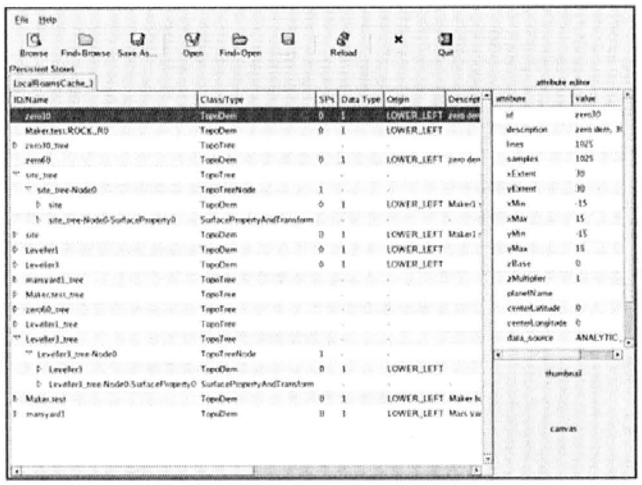
1) Objects are saved in a tree structure in the XML file with each child object saved as a leaf (node) of it's parent object's node so reconstructing an object with child objects is simply a matter of walking the tree.
2) Arrays of simple data types (integers, strings and floating point numbers) are 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 (e.g. image files) are also stored in separate files; the path to the image file is 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 the Dspace 3D graphics toolkit for visualizing the terrain models. In addition to the C++ API, **SIMSCAPE** includes a Python binding that provides a scripting interface for user scripts and run-time interaction with the models.

While **SIMSCAPE** provides a portable and well-defined C++ interface for use in applications, a Python [9] scripting interface is available for all the **SIMSCAPE** object classes. The Python interface is auto-generated using the **SWIG** [10] tool designed for this purpose. The **SIMSCAPE** Python classes closely mimic the underlying C++ classes. **SIMSCAPE**'s Python interface is very handy 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 of 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



ID/Name	Class/Type	SPs	Data Type	Origin	Descript
zero30	TopoDem	0	1	LOWER_LEFT	zero dem
Maker.test.ROCK_R0	TopoDem	0	1	LOWER_LEFT	
zero30_tree	TopoTree		-	-	
zero60	TopoDem	0	1	LOWER_LEFT	zero dem
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 -
Leveler1	TopoDem	0	1	LOWER_LEFT	
Leveler3	TopoDem	0	1	LOWER_LEFT	
marsyard1_tree	TopoTree		-	-	
Maker.test_tree	TopoTree		-	-	
zero60_tree	TopoTree		-	-	
Leveler1_tree	TopoTree		-	-	
Leveler3_tree	TopoTree		-	-	
Leveler3_tree-Node0	TopoTreeNode	1	-	-	
Leveler3	TopoDem	0	1	LOWER_LEFT	
Leveler3_tree-Node0-SurfaceProperty0	SurfacePropertyAndTransform		-	-	
Maker.test	TopoDem	0	1	LOWER_LEFT	Maker lo
marsyard1	TopoDem	0	1	LOWER_LEFT	Mars yar

attribute	value
id	zero30
description	zero dem, 30
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

of one or more **SIMSCAPE** stores. The gui also displays the defined attributes for objects in the store. 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 describe here briefly a few simulation applications that are currently using **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, 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**.

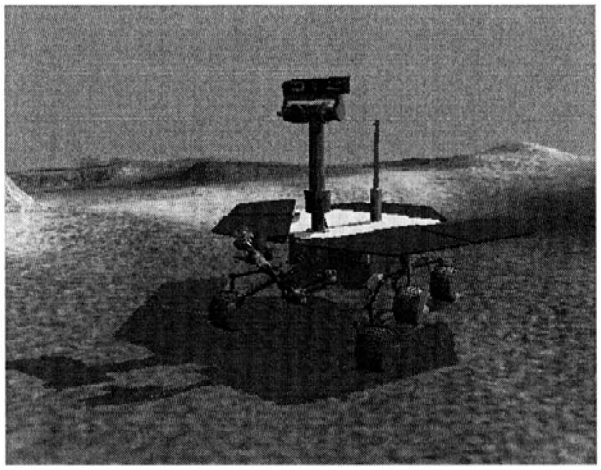


Fig. 10. Roams - Rover Vehicle Simulation Environment

- **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) is an EDL-specific extension of a JPL multi-mission simulation toolkit Darts/Dshell, which is capable of modeling spacecraft dynamics, devices, and subsystems and is in use by interplanetary and science-craft missions such as Cassini, Galileo, SIM, and Starlight. 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.

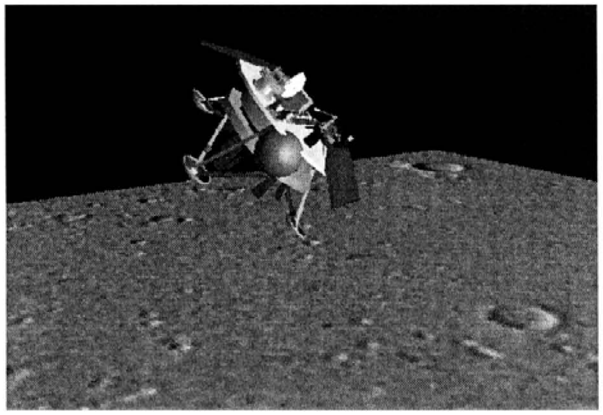


Fig. 11. DSENDS - Mars Entry Descent and Landing Simulation

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 some 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 arise within sophisticated physics-based simulations. The **SIMSCAPE** implementation has adopted open-source toolkits where avaialable 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 plugin extensions based on the ongoing use of this toolkit.

ACKNOWLEDGMENTS

We would like to thank Dr. Meemong Lee and James Wood for their many suggestions during the design and implementation of the **SIMSCAPE** software. 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.