Challenges in 3D Visualization for Mars Exploration Rover Mission Science Planning

Marsette A. Vona, III (vona@jpl.nasa.gov), Paul G. Backes, Jeffrey S. Norris, Mark W. Powell
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA

Abstract—The Science Activity Planner (SAP), currently under development by our group at the Jet Propulsion Laboratory, will be the primary tool used for science data assessment and science activity planning during the Mars Exploration Rover (MER) mission. As part of its data visualization capability, SAP interactively displays 3D terrain surface data corresponding to the MER image data products. These datasets can be very large, e.g. on the order of tens of millions of vertices for a panorama, so it is a challenge to load and display them at interactive speeds on a workstation. We describe the software techniques we are implementing to address this challenge and present recent test results. A fundamental development is the new *Visible Scalable Terrain* (ViSTa) format, a flexible and precise interchange format for terrain data. Other developments include multi-threaded asynchronous event-driven data loading, practical heuristics for geometry LOD and texture resolution selection, multi-level garbage-collector friendly data caching, and an optimized ray intersection system.

TABLE OF CONTENTS

1. INTRODUCTION

The Mars Exploration Rover mission (MER), [7], is scheduled to land two rovers on the surface of Mars in early 2004, extending the technology demonstrated in the 1997 Mars Pathfinder mission into a full "robotic field geology" system. The MER vehicles, while based on a similar 6-wheel rocker-bogey mobility platform, are larger and much more capable than the 1997 Sojourner rover. They will carry a compliment of scientific instruments called the Athena Science Payload ([6]) which includes three spectrometers, a multi-wavelength mast-mounted imager called the Pancam, a manipulator-mounted microscopic imager, plus several other instruments [8]. Additional high-resolution cameras will also be included for use in activity planning and engineering analysis.

With the exception of the microscopic imager, all MER cameras are actually stereo pairs from which both 2D and 3D datasets are acquired. This paper is focused on the software techniques we are developing to interactively visualize the 3D data, which is especially challenging because the datasets can be quite large.

The MER mission will be operated from JPL for its duration by a team of scientists selected from a number of institutions and by JPL engineers. Each Martian day, or "sol", that the rovers are active on the Martian surface, the scientists will analyze newly acquired downlink data from the MER instruments and collaborate to produce science activity plans for the subsequent sol. These science plans will then be used as input to develop the sequences of commands which are ultimately uplinked to the rovers for execution.

We are developing a software system called the Science Activity Planner (SAP), shown in Figure 1, which the mission scientists will use to analyze the 2D and 3D data from the rovers' instruments and to develop science activity plans.

The data visualization components of SAP have been entirely revamped relative to the related functionality used previously [2]. Many features have been added, but the bulk of the work has been to support high-speed loading, navigation, and manipulation of the especially large datasets that the MER instruments are expected to produce. Most MER cameras have roughly 4 times the resolution of the highest-resolution cameras that have been used on our research rovers, which were 640×480. Also, larger collections of images (typically "panoramas" taken by rotating mast-mounted cameras while the rover remains stationary) are expected to be used during MER than have been used in the past.

The increased quantity of data is especially difficult to deal with in the 3D visualization component of SAP, since here

0-7803-7651-X/03/$17.00/© 2003 IEEE
IEEEAC paper # 1181

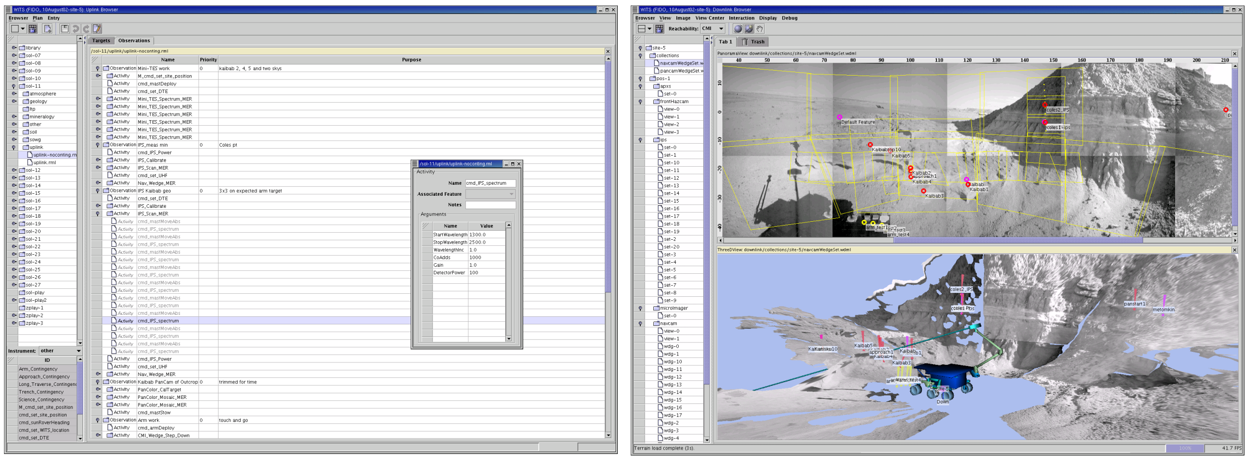


Figure 1. The Science Activity Planner (SAP) is the software system the MER mission scientists will use to analyze the 2D and 3D data from the rovers' instruments and to develop science activity plans. This figure shows a typical dual-monitor screen layout for operating SAP. The activity planning interface is on the left . The data visualization interface is on the right, where a 2D view of a dataset is shown above a 3D view of the same dataset. Data shown is from recent field test of the MER-like FIDO rover [5].

we must manipulate not only all of the 2D data (for texturing) but also the voluminous 3D spatial data which describes the structure of the imaged surfaces.

3D Data Acquisition and Processing

To illustrate the role SAP plays in 3D science data visualization we review the stages of the 3D data processing pipeline from acquisition to display in SAP. We present this review here to provide context for the bulk of our discussion in this paper, which will be focused on the software techniques we use for interactive 3D visualization. The basic steps are:

1. acquire a stereo pair of images, called a *wedge* because the camera frusta typically intersect the terrain surface near the rover in a truncated wedge shape
2. pre-process the images to compensate for optical distortion and other effects
3. perform stereo correlation to recover disparity values for as many pixels as possible
4. use the disparity values and camera calibration information to produce an array of points in 3D space called an XYZ map
5. triangulate the XYZ map to produce a surface mesh, potentially dropping some 3D points and repeating to produce multiple Levels Of Detail (LOD)
6. transform the mesh vertex coordinates to the desired output frame
7. write the mesh geometry and topology to file
8. assemble the mesh files for all wedges associated with a single camera activity (usually a panorama) into an indexed collection
9. for each wedge in the collection, load the mesh file and one of the corresponding image files (typically the left camera image is used)
10. interactively render all meshes in the collection simultaneously as triangulated surfaces textured by the corresponding images

The end result of this process, shown in Figure 2, is a visualization system in SAP which allows the MER mission scientists to virtually navigate the terrain the rover has imaged.

2. Related Work

Although it is the primary tool to be used by the mission scientists for scientific data visualization and science activity planning, SAP is not the only tool that will be used during MER operations for visualization of 3D instrument data:

- Viz ([4]) is a software package for 3D data visualization under development at NASA Ames Research Center. Viz provides some types of detailed data analysis, for example shadow simulation, that SAP does not currently support. Viz is currently limited to 3D datasets only, whereas SAP provides full capabilities to visualize both 2D (Figure 1, upper right) and 3D (Figure 1, lower right) datasets, and also allows science activity planning (Figure 1, left).
- An additional JPL-internal tool is also under development which will be used for engineering activity planning and uplink sequence validation. This tool will have its own 3D data visualization independent of SAP and Viz.

Another significant area of related work is the relatively large body of research in multiresolution data visualization, especially that which is specific to terrain data rendering. One well-known and representative system for terrain rendering is ROAM ([9]), which stands for "Real-time Optimally Adapting Meshes." ROAM is an advanced and relatively complex system which manipulates a triangulated terrain mesh in real time on a per-triangle basis to maximize framerate while attempting to optimize certain quality-motivated error metrics.

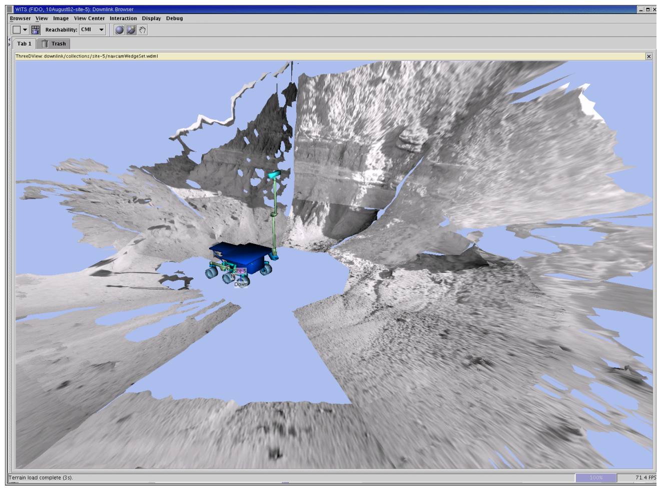


Figure 2. A major component of SAP is a 3D visualization system which allows the MER mission scientists to virtually navigate the terrain the rover has imaged. This figure shows a "panorama" of terrain "wedges". Each wedge was acquired by a rotating mast-mounted camera, shown deployed in this image (colored cyan in the rover model). Areas in which 3D data recovery has failed (typically due to failure of stereo correlation) are left as holes in the terrain model through which the light-blue background is visible. One entire wedge is absent in the foreground, likely due to an acquisition or processing error. Data shown is from recent field test of the MER-like FIDO rover [5].

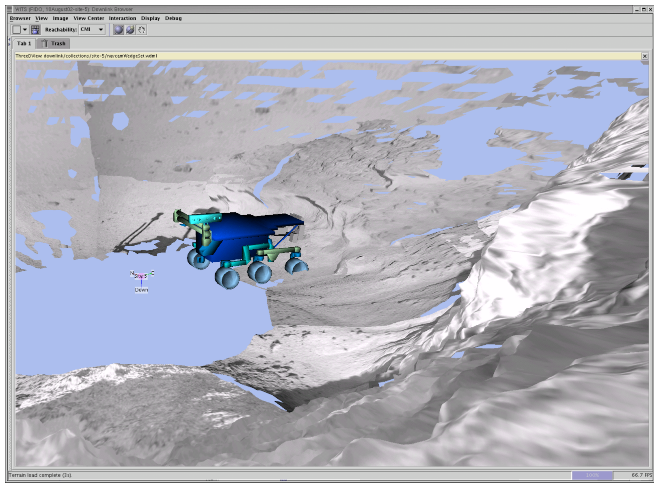


Figure 3. Overhangs are sometimes critically important features of terrains visualized in SAP. This figure shows an actual situation that occurred during a recent test of the MER-like FIDO rover [5]. The rover has approached an outcropping (visible to the right of the rover in this view) which includes some overhanging geometry. In subsequent operations we attempted to deploy the rover's instrument arm onto the outcropping.

One major issue with ROAM is that, in its usual implementation, it supports only 2.5D terrains (i.e. elevation maps). As shown in Figure 3, overhangs (which cannot be represented by a ground-aligned elevation map) are sometimes critically important features of terrains visualized in SAP. During initial design, we also felt that the real time per-triangle computations of ROAM, and of other systems like it, were potentially overkill in SAP. As shown in Figure 2, the datasets we visualize are naturally segmented by virtue of the fact that they were acquired in wedges from multiple pairs of stereo images, each with a relatively narrow field of view. We predicted that a system based on discrete, fixed levels of detail for each wedge would be relatively simple to implement and would still provide adequate performance and quality. This prediction has proved accurate thus far (c.f. Sections 4 and 5).

3. Visible Scalable Terrain (ViSTa) Format

As we described in Section 1, the result of the terrain processing pipeline for MER is one or more files containing triangulated terrain surface data. The format of the subset of these files that are read by SAP was carefully designed, as it has a large effect on SAP's data loading performance, and a major impact on the network bandwidth and disk space required for data distribution and storage. In other implementations an ad-hoc terrain file format was used which was the source of several major inefficiencies. The larger datasets expected to be produced by MER demanded that we revisit this format. We evaluated a number of existing formats, as described below, and eventually decided that the deficiencies of each warranted the design of a new format.

We have named the new format ViSTa, which stands for "Visible Scalable Terrain", because it is specifically adapted to terrain data acquired from stereo vision hardware, and it also is designed to support data and performance scalability. We have endeavored to design ViSTa so that it is re-usable in any system which visualizes terrain acquired from stereo vision hardware. In the remainder of this Section we will give only an overview of ViSTa. The full specification document, [1], should always be thoroughly consulted when developing software that reads or writes ViSTa.

Problems with Existing Formats

Clearly, if a data format had already existed that was well-documented, available for our use, and which met all of our needs, then we would have seriously considered using it rather than develop our own new format. However, as we surveyed the existing formats, we discovered that each had one or more major deficiencies with respect to our intended use in SAP.

We do not claim that our survey was exhaustive. It is merely what we were capable of performing given our budget and time constraints. We do believe we have covered nearly all major geometry formats for which full technical documentation is readily and publicly available on the internet.

A number of common terrain data formats are inherently limited to 2.5D (i.e. elevation map) models. This is reasonable given their intended uses, which are typically GIS, military/commercial simulation, and games. However, as Figure 3 illustrates, overhangs are sometimes significant and critically important features in terrain datasets used in SAP. Thus we must omit from consideration all data formats that are strictly 2.5D. These include United States Geological Survey (USGS) Digital Elevation Model (DEM), USGS Digital Terrain Elevation Data (DTED), USGS National Elevation Dataset (NED), and Virtual Terrain Project (VTP) Binary Terrain (BT) format.

Of course there also exist many fully three-dimensional geometry data formats. Mostly these have been developed for use in CAD, virtual reality, computer art/animation, games, and mathematical visualization applications. Terrain data can be represented as a special case in any of these.

A few of the 3D formats do not include provisions for storing texture mapping[1] information. As we describe in the *Visibility* section below, this information is important in SAP because it allows the terrain to be colored by its actual sensed visual appearance (and other co-registered data), which greatly enhances the usefulness of the visualization for scientists. 3D formats which do not support texture mapping include Autodesk® Data eXchange Format (DXF) and STereoLithography (STL).

Many 3D formats do include texture mapping information. However, of these, most have no provisions for data and/or performance scalability, which typically involves including information to construct lower-resolution versions of the full model. In the *Scalability* section below, we elaborate on the scalability we desire for SAP's terrain data. Formats which we omit due to lack of scalability support include LightWave® Scene (LWS), Alias|Wavefront® OBJect (OBJ), Visualization ToolKit (VTK), and 3ds max[TM] (3DS).

Only a few common 3D formats support both texture mapping and scalability. Two of these, SGI® Open Inventor[TM] binary format and SGI® OpenGL Performer[TM] Performer Binary Format (PFB), are proprietary. The third, Virtual Reality Modeling Language (VRML), is popular and well-documented. However, it is an ASCII format, and for large datasets we felt that this would lead to greatly inflated file sizes and load times relative to a binary format.

Design Goals

We developed some general design goals for ViSTa in addition to addressing the specific deficiencies of the various existing formats called out above:

- data accuracy, especially with respect to the abilities to display and pick accurate locations on the terrain and to accurately display overlaid representations of auxiliary science data
- minimization of file size
- platform portability
- run-time efficiency in computation and memory usage for systems which create, process, and display ViSTa format terrains
- support of both high-resolution/resource-intensive and lower-resolution/less-resource-intensive (e.g. public outreach) applications
- support of both single-wedge (i.e. all vertices visible from the left image of a single stereo pair) and merged multi-wedge terrains
- ease of generation from stereo vision data
- ease of integration with MER subsystems which read and write terrain data

ViSTa Format Basics

ViSTa is a binary format composed of fixed-length fields in a well-defined sequence. It is specifically designed for the visualization of textured terrain surface meshes acquired from stereo vision hardware. Geometric data is stored using IEEE 754 32-bit floating point numbers as this is commonly required by modern workstation rendering systems. Because the basic ViSTa structure is a contiguous array of vertices combined with a set of indexed triangle strips, it should be possible on many modern workstations to load the data from disk in blocks and send it to the rendering system without any per-vertex or per-face loops in the front-end application code whatsoever. As we describe below in Section 4, we have achieved this in our implementation of SAP.

All spatial geometric data in a ViSTa file are specified in units of meters. All vertices are specified in the same coordinate system. The specification of this coordinate system is implementation-dependent, but in all cases must be Cartesian and right-handed. A fixed space is reserved in every ViSTa file for implementation-dependent specification of coordinate system (this space may be unused in some implementations, e.g. if the coordinate system in that implementation is implicit).

To keep the ViSTa format flexible and re-usable, we have chosen to explicitly leave the syntax and semantics of some sections of a ViSTa file implementation-dependent. Specific ViSTa implementations are defined by documents which specify how each of these sections are to be interpreted. Each implementation is assigned a 4-byte implementation identifier. A system which creates a ViSTa file must specify the implementation it is using by writing the corresponding implementation identifier in the header section of the file. Systems which read ViSTa files must check this field and interpret the data according to the indicated implementation, or display an

[1] *Texture mapping is a process by which a 2D image is painted onto the faces of a 3D model. It is typically accomplished on modern workstation graphics hardware by associating a pair of normalized texture coordinates $(s, t) \in [0, 1]^2$ with each vertex. These texture coordinates proportionally indicate the position of the corresponding vertex in the 2D texture image, and interpolating between them gives the texture coordinates for all points on each face.*

error message if they do not support the implementation.

So far we have defined three ViSTa implementations: SMP, FDO, and MER. SMP is a “Simple” implementation that we employed for early development and testing. FDO is the implementation we employed for the FIDO rover field test described below in Section 5. MER is the implementation we are developing for the MER mission.

Bounding Boxes

A ViSTa file includes a direct representation of the axis-aligned bounding box that encloses the entire terrain in the file, as well as per-texture bounding boxes at each LOD. Bounding boxes are very useful because the can be used by rendering systems to do high-speed visibility culling for improved rendering performance. The Java 3D[TM] rendering system that we use in SAP has this feature. We also use the bounding boxes in our ray intersection implementation as described below in Section 4. Since the bounding boxes are fixed it is more efficient to compute and store them once when the ViSTa file is created, rather than have to generate them from the terrain data every time the file is loaded.

ViSTa Layout

In this section we give an overview of the layout of a ViSTa file. However as some details of the ViSTa specification are still in flux at the time of this writing, we do not provide the lowest-level details. The ViSTa specification, maintained at [1], defines the syntax and semantics of every field at the bit level.

A ViSTa file is an ordered sequence of fixed-length `field`s which follows this grammar:

ViSTa := `VSTHeader`
`BoundingBox`
`TextureRef`^{T}
`CoordinateSystem`
`Vertex`$^+$
LOD$^+$

LOD := `LODHeader`
`BoundingBox`^{T}
Patch$^+$

Patch := `PatchHeader`
`IndexArrayLength`^{n}
IndexArray^n

IndexArray := `Index`$^+$

The `VSTHeader` field contains basic information about the ViSTa file including the ViSTa format version to which the file corresponds, the ViSTa implementation to which it adheres, byte order, total number of texture references, total number of vertices, and total number of LOD.

The `BoundingBox` field that follows the `VSTHeader` specifies the corners of an axis-aligned bounding box which contains the entire terrain surface defined in the file.

A `Vertex` field contains the spatial (x, y, z) coordinates of a vertex on the terrain surface and the normalized texture coordinates $(s, t) \in [0, 1]^2$ at that vertex as IEEE 754 32-bit floating point numbers.

The `Vertex`$^+$ section is a pool of vertices shared by all LOD. `Vertex` fields in this section are implicitly assigned zero-based integer indices according to their position in the file.

A `TextureRef` is an implementation-specific reference to a texture image. All texture images are stored outside the ViSTa file. Each `TextureRef` is implicitly assigned a zero-based integer index according to its position in the file. ViSTa files containing data from only a single stereo pair normally contain exactly one `TextureRef`. In the MER implementation, this field is a relative path to a file containing a left-camera acquired image.

The `CoordinateSystem` field is an implementation-specific area where data may be stored to locate the terrain defined in the file relative to other terrains and/or to externally known coordinate frames. Please refer to the ViSTa specification, [1], for details on this field.

Multiple LODs within a single ViSTa file are presented in increasing order from least-detailed to most-detailed. Each LOD is implicitly assigned a zero-based integer index according to its position in the file.

An LOD is composed of a header, a set of axis-aligned `BoundingBox`es, one per texture, and a set of Patches.

A Patch is a collection of indices into the `Vertex` pool that defines the topology of a chunk of the terrain surface at a given LOD which is textured entirely by the image corresponding to one `TextureRef`. There are two types of patches: point cloud and triangle strip. A point cloud patch contains a set of zero-dimensional points and is typically used only when triangulation is not available or is infeasible. A triangle strip patch contains a set of triangle strips which define a localized region of the terrain surface. Each patch contains a header that identifies its type and the index of the `TextureRef` to which it corresponds.

Visibility

By placing triangles in Patches which each reference a specific texture, a ViSTa file specifies an association for each triangle in the terrain to a specific texture image such that

- the texture image is (usually the rectified[2] version of) an

[2] I.e. corrected for nonlinear optical distortion.

image acquired by the left camera of the stereo vision hardware, or a processed version thereof

- the triangle is *visible* in the texture image, where the definition of *visible* is that
  1. the triangle is not occluded in any part in the texture image by any triangle in the same LOD
  2. the geometric back-projection of (front face of) the triangle into the image plane (see [10]) is entirely within the bounds of the image

In addition to these rules for vertex visibility, SAP imposes additional constraints on the accuracy of the texture coordinates (s, t) that are associated with every vertex in a ViSTa file.

Texture coordinates for a vertex are *accurate* if they define a point in the texture image plane inside the pixel containing the geometric back-projection point of the vertex. This definition can also be viewed as an algorithm to generate texture coordinates for an arbitrary vertex, provided that a texture image in which the vertex is visible is already known. Additionally, such texture coordinates are available as a trivial by-product of the stereo reconstruction algorithm which performs the forward mapping of image pixels to 3D vertices: if such an algorithm maps a pixel (i, j) of the (rectified) left image to 3D vertex $\mathbf{v}$, and the (rectified) left image is N pixels wide and M pixels tall, then the normalized texture coordinates associated with $\mathbf{v}$ are $(s, t) = (i/N, j/M)$.

This definition of texture coordinate accuracy ensures that visual features in the texture image are properly co-registered with the geometry of the terrain, which is important in SAP because the image is typically rich in details that scientists use to perform visual localization, select points of interest, and plan activities. As we describe below in Sections 4 and 6, we can also leverage the texture coordinates' accuracy to implement picking, overlay graphics, and auxiliary science data co-registration.

Scalability

ViSTa supports data scalability by allowing the definition, within a single file, of terrains ranging in size from a fraction of the data from a single stereo pair to a full panorama of stereo pairs or more. Additionally, SAP supports loading multiple ViSTa files into the same interactive scene, as shown in Figure 2.

The geometry in a ViSTa file is specified as an ordered set of Levels Of Detail (LOD) to support scalable visualization performance and resource usage. Each LOD has a header field which contains a distance threshold below which SAP will consider switching to the next higher LOD, if available. The exact semantics of this field are described below in Section 4.

The header field for each LOD identifies the highest-indexed vertex referenced by that LOD. This feature aids the implementation of a simple utility for reducing the available number of LOD (and hence the total size) of a ViSTa file. Such a utility will likely be useful for producing smaller datasets suitable for internet distribution for public outreach.

Limitations

In its current form, the ViSTa format does have certain limitations. It has been designed with only terrain data acquired from stereo vision hardware in mind. There are actually no serious restrictions at the bit-level that would make it difficult to generalize ViSTa to terrain data acquired from many other sources. Some of the semantic requirements defined in the ViSTa specification ([1]) only seem applicable to vision-acquired terrains and would likely have to be relaxed for terrains acquired through other means.

As it is essentially a surface mesh format, ViSTa is appropriate for visualization purposes. It may not be as directly useful for geometric interrogation and analysis applications, where occupancy-grid based representations are sometimes preferred.

4. Implementation Details

We now turn to some of the details of SAP's implementation, which illustrate how the design of the ViSTa terrain format can be leveraged to build a high-performance interactive visualization system.

The 3D visualization component in SAP was developed relatively quickly, with the attention of only one full-time software engineer for about 10 months. Of course, this would not have been possible without the collaborative support of the entire SAP development group (which itself has included only 3 full-time developers).

Like the rest of SAP, the 3D visualization component is developed entirely in the Java[TM] language on RedHat® Linux workstations. We use Sun Microsystems' Java[TM] 1.4 development environment, combined with the Sun Java 3D[TM] 1.3.0 (ported to Linux by the Blackdown project) package and the Sun Java Advanced Imaging (JAI) 1.1.1 package.

Level Of Detail (LOD) Switching

In SAP, the geometric LOD and texture resolution for each terrain wedge in the scene are possibly modified each time the viewpoint is moved during interactive navigation. In our current implementation these computations are based on the distance from the viewpoint Center-of-Projection (CoP) to the centroid of the bounding box of each terrain wedge (recall that this bounding box is present in the ViSTa file for the wedge, so computing these centroids is a constant-time operation per wedge).

As shown in Figure 4, the ViSTa format wedges each define a set of discrete geometric LOD. Each LOD in the ViSTa file has a header which includes a field called the "LOD switch threshold". This is the distance from viewpoint CoP to the

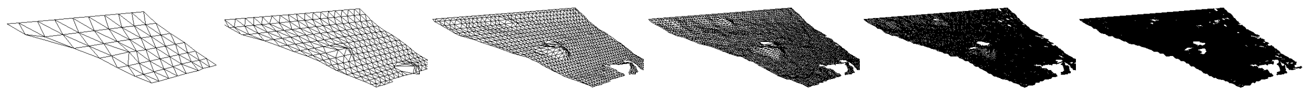


Figure 4. SAP loads 3D terrain data from per-wedge ViSTa files which each contain a set of discrete Levels Of Detail (LOD). Shown here are the 6 LODs in a ViSTa file for one wedge, which range in size from about 100 triangles to about 100,000 triangles. Data shown is from recent field test of the MER-like FIDO rover [5].

wedge centroid below which a more detailed LOD should be displayed. These threshold values are currently computed by the following heuristic algorithm (contributed by Jack Morrison):

For each LOD i:
1. Compute w_i, an estimate of the average "width" of a triangle in LOD i, according to the following formula

$$w_i = \frac{\text{(average terrain bounding box side)}}{\sqrt{\text{number of triangles in LOD } i}} \quad (1)$$

2. Compute t_i, the LOD switch threshold for LOD i, according to the following formula, which makes t_i the distance at which a triangle of "width" w_i subtends a viewing angle of $1°$:

$$t_i = \frac{0.5 w_i}{\tan 0.5°} \quad (2)$$

Or, use the approximation:

$$t_i \approx 57 w_i \quad (3)$$

SAP loads texture image data separately from the terrain geometry. Each wedge is textured by (a possibly processed version of) the left image of the stereo pair from which it was generated. The resolutions of the images actually sent to the rendering system are generally not the same as the original resolutions of the images; modern rendering systems require texture dimensions to be powers of 2. SAP further downsamples the images by an additional power of 2 in each dimension to produce a smaller image that still looks good, which works because the farther away the viewpoint is from a wedge the smaller the wedge appears. SAP currently uses a simple heuristic to select the appropriate resolution for the texture on each wedge: First, a bounding sphere is fit to the wedge geometry (this can be done in constant time by operating only on the wedge bounding box). Next, the bounding sphere is conceptually projected from its location in the scene onto the current viewpoint canvas, resulting in a circle. The radius of the circle is then measured in pixels, and the largest available texture resolution whose maximum dimension is less than or equal to the measured radius is selected.

SAP aggressively attempts to use the least-detailed LOD and smallest texture resolution possible for all wedges according to the current viewpoint location. Furthermore, as we describe next, LOD levels and textures that are not currently in use may be culled from memory as necessary.

Multi-Level Caching

SAP employs object caching in several places to enable high-speed loading of previously viewed data and to provide a central location for referencing large data objects to avoid making copies. We have developed a generic Java[TM] object cache which works with the Java Garbage Collector (GC) to ensure that unreferenced objects are only flushed from the cache when memory is actually running low. As Figure 5 illustrates, we employ several instances of this object cache in our terrain system.

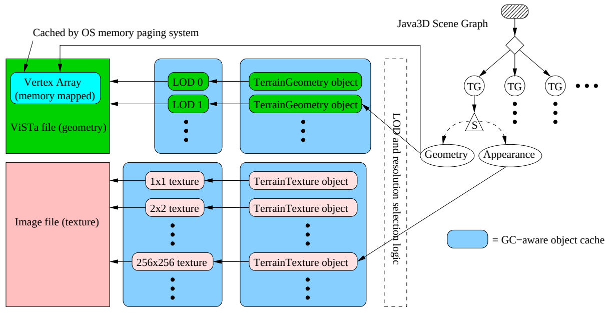


Figure 5. We have developed a generic Java[TM] object cache which works with the Java Garbage Collector (GC) to ensure that unreferenced objects are only flushed from the cache when memory is actually running low. This figure shows the uses of instances of this object cache in our terrain system. An additional level of caching is achieved by memory mapping the vertex array in the ViSTa file.

As the figure shows, we cache the topology data for each LOD separately, a feature easily implemented using the discrete LODs available in the ViSTa file. We also implement an additional level of caching by memory-mapping the vertex array from the ViSTa file. This not only speeds loading by avoiding copies of this voluminous data structure, but it also transparently provides demand loading and caching of the data by taking advantage of the OS memory paging system. This works because the ViSTa specification encourages ViSTa file writers to order the vertex array so that vertices used in an LOD appear contiguously, so that those vertices that are used by higher (more detailed) LOD appear later in the array, and so that more detailed LOD share the vertices used by less-detailed LOD. Together, these ensure locality of reference for the vertex data, which makes page-caching useful.

Asynchronous Data Loading

In our implementation of SAP we take advantage of Java[TM]'s threading system to load multiple parts of a data collection concurrently, while still allowing the user to interact with the scene. For very large datasets, e.g. the one described below in Section 5, this has the great advantage that the user does not have to wait for the entire dataset to load before starting to investigate it.

We implement asynchronous data loading by submitting tasks to instances of a generic thread pool. The thread pool is implemented as a constant number of worker threads and a queue to which tasks are submitted for execution. Tasks are pulled from the queue and assigned to worker threads as the threads become available. When loading 3D datasets we submit each LOD fetch and each texture fetch as a separate task to a geometry loading thread pool and to a texture loading thread pool, respectively.

Ray Intersection

Ray intersection is another feature in SAP which is simplified and enhanced by ViSTa features. This is the problem of taking an arbitrary spatial ray in the 3D scene and finding the ordered set of points that it intersects on the (outward facing) terrain surface. Ray intersection is currently used in SAP to implement interactive picking of points on the terrain and to produce accurate simulations of planned remote sensing operations. For example, if a scientist wants to aim a mast-mounted line-of-sight remote spectrometer instrument on a specific rock, we simulate the activity by computing rays around the perimeter of the spectrometer's (typically very narrow) view frustum. We use our ray intersection system to find the nearest points where the rays intersect the previously sensed terrain (normally the area of interest, if it is on the terrain at all, has been previously imaged). Once we have the intersection points we can project them back to screen space and connect them to form a polygon which represents the spot on the terrain that the spectrometer will likely measure. We call this simulation feature a "footprint," and it provides valuable feedback to the scientists about the probable effects of their planned activities. As we mention below in Section 6, we have not finished implementing the footprint feature in our 3D views, but it has been completed in the 2D views (which still rely on ray intersection against the terrain models behind-the-scenes).

Before we describe SAP's ray intersection algorithm we need to describe how SAP implements interactive picking in 2D views, as the same functionality is leveraged by a part of the ray intersection algorithm. When the user clicks on a point in an image in a 2D view, we display the image pixel coordinates of the clicked point and also the spatial (x, y, z) coordinates of the point that were recovered from stereo correlation, if any. To produce this latter display, SAP performs a lookup in a pre-computed *range map*, which is normally an amortized constant-time accessible cache of all the stereo correlation results for the wedge corresponding to the image.

We now present the ray intersection algorithm. Given a ray $\mathcal{R} = (\mathbf{p}_{\text{start}}, \hat{\mathbf{v}}_{\text{direction}})$ and a set $\mathcal{T}$ of terrains:

```
1. for each terrain wedge $w \in \mathcal{T}$:
  (a) if $\mathcal{R}$ intersects the bounding box for $w$ specified in its ViSTa file then:
    i. Request the topology information for the lowest (i.e. least-detailed) LOD of $w$ from the centralized cache described in the previous section.
    ii. Using the lowest-LOD topology information and the memory-mapped pool of vertices from the ViSTa file for $w$, iterate over all the triangles in the lowest LOD of $w$, checking the intersection of $\mathcal{R}$ with each. Add any front-face intersections found to the result list. Compute both the spatial and normalized texture coordinates for each intersection point by interpolating the vertex data for the intersected triangle.
2. Compute the texture image pixel coordinates for each intersection point by scaling the normalized texture coordinates to the dimensions of the corresponding texture image.
3. Sort the result list in increasing order by distance from $\mathbf{p}_{\text{start}}$.
```

The result of this algorithm is an ordered list containing information about each intersection point. At this stage the normalized texture coordinates and the texture image pixel coordinates have already been computed. Spatial intersection coordinates are also available, but they are computed only against lowest-resolution LOD of the terrain surface. Sometimes more accurate spatial coordinates are not required, but if necessary, we can find high-resolution spatial coordinates quickly by performing a range map lookup using the texture image pixel coordinates in the same way as we do for a pick in a 2D view. If this lookup fails (i.e. because stereo correlation failed for the pixel in question) then this is a strong suggestion to SAP that the spatial coordinates of the point are uncertain and should not be trusted.

The ray intersection algorithm leverages ViSTa features in several ways. First, it uses the terrain bounding box to perform a quick intersection check. Second, it only iterates over the lowest LOD of a terrain, which can have 3 or more orders of magnitude fewer triangles than the highest LOD (c.f. Figure 4). Third, it uses the guaranteed semantics and accuracy of the texture coordinates to compute the texture image pixel coordinates at the intersection point, which SAP can use directly in some cases, and from which SAP can quickly look up a high-resolution spatial intersection point when necessary.

5. Test Results

SAP was recently tested aggressively in an intense 10-day field test of the MER-like FIDO rover ([5]), in which the rover was remotely operated in a Southwest desert location. Most of the data shown in previous figures was actually acquired, analyzed, and used for science operations planning during this test. The operations center for this test, shown in Figure 6 was staged at JPL.

The 3D visualization capabilities of SAP were used extensively in this field test, which included some relatively large datasets. The largest single 3D dataset acquired during the test was a panorama taken from the initial rover location. This dataset consists of 265 terrain wedges, each with an associated 640x480x24 texture image. Aggregately, the highest LOD and texture resolution of the wedges in this set contained over 14 million triangles (17 million vertices) and over 232MB of texture data. Running under RedHat® Linux 7.2 on PC workstations with dual Intel® Xeon[TM] 1.7GHz processors, 1GB memory, NVIDIA® Quadro[TM] 4 750 XGL graphics systems, and high-speed IDE disks, SAP reliably loaded this dataset and allowed interactive navigation at about 20 frames per second, even under high machine load. Load times (when data is not already in the OS disk cache) are about 20s until first interactive geometry is available, about 35s until all geometry is loaded, and about 53s until all textures are loaded.

6. Future Work

Even though SAP was comprehensively tested in the recent FIDO field test, it is not yet complete. Time- and budget-permitting, we intend to consider several substantial improvements to the 3D visualization component.

One improvement we may add is the ability to display simulated camera activity *footprints* on the 3D terrain surface. A footprint in this usage is the outline of the intersection of a simulated camera view frustum with the terrain surface, which gives scientists a sense of the image that will result from a planned camera activity. We already generate footprints in the SAP 2D views, as shown in the upper right half of Figure 1.

Another addition we are planning is the ability to overlay auxiliary 2D datasets onto the textures in a 3D view. This could allow the scientists to see a representation of spectral measurement data directly on the terrain feature from which the data was acquired. This feature is described in detail in [11].

We predict that the well-defined semantics of texture coordinates in ViSTa will aid us in implementing each of these features.

7. Conclusions

We have described the design and implementation of the 3D visualization system in SAP, the primary science data analysis and science activity planning tool for the Mars Exploration Rover mission, currently under development.

A major challenge that we faced in the development of SAP was the definition of the format that it would use to represent 3D terrain data. After analyzing many of the popular formats currently in use, we decided that a new format was needed to suit our requirements. To this end we developed the Visible Scalable Terrain format, or ViSTa. ViSTa is a full 3D format (i.e. not just a 2.5D elevation map). ViSTa includes bounding boxes, texture coordinates with well-specified semantics to allow the accurate overlay of 2D data onto the 3D terrain surface, and it also includes Levels of Detail to enable scalable performance and resource usage.

The design of the ViSTa format has enabled us to build some advanced features into SAP, including LOD switching, multi-level caching, asynchronous data loading, and optimized ray intersection. Aggregately these features give SAP the capacity it needs to load and manipulate large datasets on PC-class workstations, and to this end we gave concrete results for SAP's measured performance on a specified machine configuration during a recent mission-like field test.

8. Acknowledgments

We thank Jack Morrison for implementing the ViSTa output generator for the FIDO and MER terrain processing pipelines and for many helpful conversations about the design of the ViSTa format, the FIDO rover engineering team for working closely with us in developing and testing our tool, and Greg Tharp for implementing the low-level triangle intersection routine which is at the heart of our ray intersection system.

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] Marsette A. Vona, Mark W. Powell, "Visible Scalable Terrain (ViSTa) Format," JPL document D-22924.

[2] Paul G. Backes, Kam S. Tso, Jeffrey S. Norris, Gregory K. Tharp, Jeffrey T. Slostad, Robert G. Bonitz, and Khaled S. Ali, "Internet-Based Operations for the Mars Polar Lander Mission," *Proceedings of the IEEE Conference on Robotics and Automation*, San Francisco, CA, pp. 2025–2032, April 2000.

[3] Paul G. Backes, Jeffrey S. Norris, Mark W. Powell, Marsette A. Vona, Robert C. Steinke, Justin V. Wick, "Science Activity Planning for the Mars Exploration Rover Mission", in submission to *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, March 2003.

[4] Laurent A. Nguyen, Maria Bualat, Laurence J. Edwards, Lorenzo Flueckiger, Charles Neveu Kurt Schwehr, Michael D. Wagner, and Eric Zbinden, "Virtual Reality Interfaces for Visualization and Control of Remote Vehicles", *Autonomous Robots*, **11**, pp. 59–68, 2001.

[5] P. S. Schenker, E. T. Baumgartner, L. I. Dorsky, P.

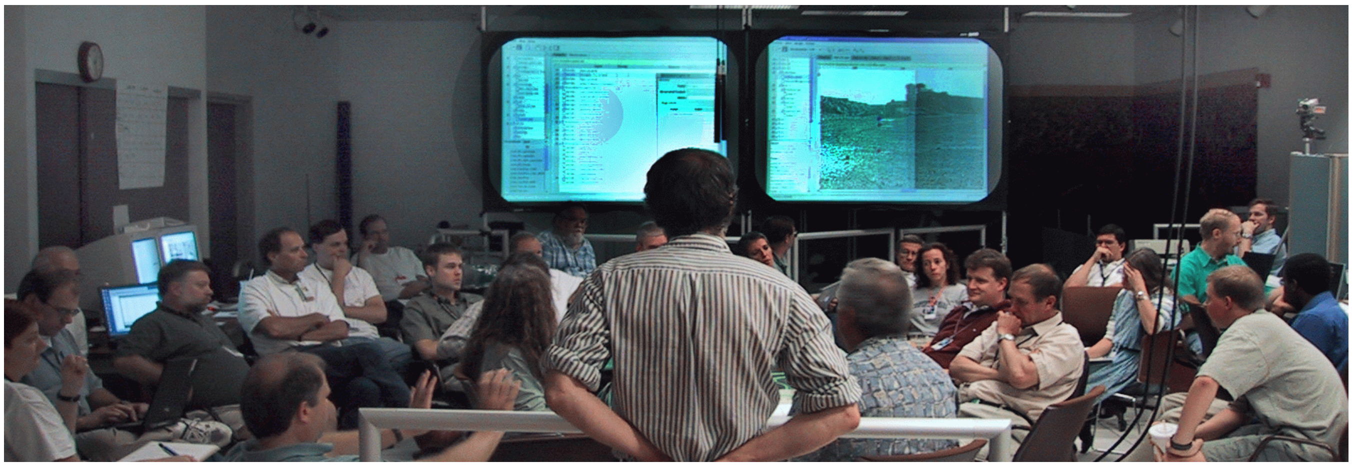


Figure 6. SAP was recently tested aggressively in an intense 10-day field test of the MER-like FIDO rover ([5]), in which the rover was remotely operated in a Southwest desert location. The operations center for this test, shown here, was staged at JPL.

G. Backes, H. Aghazarian, J. S. Norris, T. L. Huntsberger, Y. Cheng, A. Trebi-Ollennu, M. S. Garrett, B. A. Kennedy, A. J. Ganino, R. E. Arvidson, and S. W. Squyres, "FIDO: A Field Integrated Design & Operations Rover for Mars Surface Exploration," *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-'01)*, Montreal, Canada, June 2001.

[6] The Athena Science Payload public website at Cornell. http://athena.cornell.edu.

[7] The Mars Exploration Rover public website at the Jet Propulsion Laboratory. http://mars.jpl.nasa.gov/mer/.

[8] Office of Space Science, National Aeronautics and Space Administration, "Announcement of Opportunity Mars Exploration Rover Mission Participating Scientist Program," Announcement of Opportunity 01-OSS-04, Released August 29, 2001.

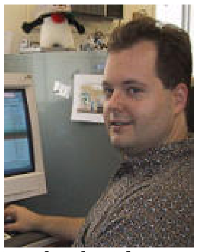
[9] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein, "ROAMing Terrain: Real-Time Optimally Adapting Meshes," *Proceedings of IEEE Visualization*, pp. 81–88, 1997.

[10] Y. Yakimovsky and R. Cunningham, "A System for Extracting Three-Dimensional Measurements from a Stereo Pair of TV Cameras," *Computer Graphics and Image Processing*, **7**, pp. 195–210, 1978.

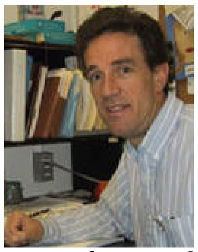
[11] Mark W. Powell, Jeffrey S. Norris, and Paul G. Backes, "Visualization of Coregistered Imagery for Remote Surface Operations," in submission to *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, March 2003.

Marsette A. Vona, III *Marsette Vona is a computer scientist, software engineer, and electromechanical hardware developer. He is a member of the technical staff of the Mobility Systems Concept Development Section at the Jet Propulsion Laboratory. At JPL, his work is currently focused in the areas of high-performance interactive 3D data visualization for planetary exploration, user-interface design for science data analysis software, and Java software architecture for large, resource-intensive applications. Marsette received a B.A. in 1999 from Dartmouth College in Computer Science and Engineering, where he developed new types of hardware and algorithms for self-reconfigurable modular robots. He completed his M.S. in 2001 at MIT's Precision Motion Control lab, where he developed a high-resolution interferometric metrology system for a new type of robotic grinding machine. Marsette was awarded the Computing Research Association Outstanding Undergraduate Award in 1999 for his research in Self-Reconfigurable Robotics.*

Jeffrey S. Norris *Jeff Norris is a computer scientist and member of the technical staff of the Mobility Systems Concept Development Section at the Jet Propulsion Laboratory. At JPL, his work is focused in the areas of distributed operations for Mars rovers and landers, secure data distribution, and science data visualization. Currently, he is a software engineer on the Mars Exploration Rover ground data systems and mission operation systems teams. Jeff received his Bachelor's and Master's degrees in Electrical Engineering and Computer Science from MIT. While an undergraduate, he worked at the MIT Media Laboratory on data visualization and media transport* *protocols. He completed his Master's thesis on face detection and recognition at the MIT Artificial Intelligence Laboratory. He lives with his wife, Kamala, in La Crescenta, California.*



Mark W. Powell *Mark Powell is a member of the technical staff in the Mobility Systems Concept Development Section at the Jet Propulsion Laboratory, Pasadena, CA since 2001. He received his B.S.C.S. in 1992, M.S.C.S in 1997, and Ph.D. in Computer Science and Engineering in 2000 from the University of South Florida, Tampa. His dissertation work was in the area of advanced illumination modeling, color and range image processing applied to robotics and medical imaging. At JPL his area of focus is science data visualization and science planning for telerobotics. He is currently serving as a software and systems engineer, contributing to the development of science planning software for the 2003 Mars Exploration Rovers and the JPL Mars Technology Program Field Integrated Design and Operations (FIDO) rover task. He, his wife Nina, and daughters Gwendolyn and Jacquelyn live in Tujunga, CA.*



Paul G. Backes *Paul Backes is a technical group leader in the Mobility Systems Concept Development section at the Jet Propulsion Laboratory, Pasadena, CA, where he has been since 1987. He received the BSME degree from U.C. Berkeley in 1982, and MSME in 1984 and Ph.D. in 1987 in Mechanical Engineering from Purdue University. He is currently responsible for distributed operations research for Mars lander and rover missions at JPL. Dr. Backes received the 1993 NASA Exceptional Engineering Achievement Medal for his contributions to space telerobotics (one of thirteen throughout NASA), 1993 Space Station Award of Merit, Best Paper Award at the 1994 World Automation Congress, 1995 JPL Technology and Applications Program Exceptional Service Award, 1998 JPL Award for Excellence and 1998 Sole Runner-up NASA Software of the Year Award. He has served as an Associate Editor of the IEEE Robotics and Automation Society Magazine. Paul initiated the development of WITS/SAP and is the team leader.*