An Efficient Modularized Database Structure for a High-resolution Column-gridded Mars Global Terrain Database

J. Bradley
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada

Abstract This paper discusses a modularized design for a Mars Global Terrain Database. The design provides for elevation data with respect to a triaxial ellipsoidal reference datum developed for Mars by USGS. Terrain data is recorded for 1-second of arc almost square grid elements over the surface of Mars. A 2000-Gigabyte column-gridded relation called Terrain contains the surface terrain data. Data for Terrain is expected in 1999-2000 from the Mars Global Surveyor satellite currently in initial polar orbit around Mars. Each tuple of Terrain contains data for a N-S column-grid of 900 1-second grid elements. There is thus a set of tuples per 1-degree rectangle, with the number of tuples per set decreasing with the cosine of latitude. Surface resolution is 16.5 meters or better. The design constrains tuple sizes in Terrain to permit efficient blocking and manipulation of the records of the underlying storage file. Terrain contains a virtual-attribute function for geodetic computations relating to the triaxial ellipsoidal reference datum.

The database also relates Mars feature-type relations to Terrain. Terrain's gridded structure is transparent to users writing SQL expressions to retrieve Terrain data on the basis of specific features. Many different distinct feature-type relations can be included. At least two of these participate in recursive relationships. The design also allows attachment of additional feature-type relations in a modular manner, correctly related to Terrain, without affecting the contents of Terrain. The design is intended to enable efficient exploration of the planet at all levels of scale.

Key words database, elevation, grid, Mars, terrain,

1. Introduction

During the last two decades there has been increased use of digital terrain databases [1, 2], often obtained by remote sensing [3, 4], of the Earth's surface. But now, with the arrival of the Mars Global Surveyor (MGS) spacecraft in Mars polar orbit in the fall of 1997, a new digital terrain database application possibility is emerging, for exploration of the terrain and geology [5] of the planet Mars. This MGS spacecraft should return the data necessary for a high resolution Mars Global Terrain Database whose primary function would be the facilitation of Mars terrain exploration.

The MGS spacecraft is equipped with a laser altimeter [6] for elevation measurements, and cameras with resolution down to a few meters, as well as other scientific instruments. All data returned from the spacecraft will be digital - digital pixel in the case of photogrammetric data. Remote-sensed photogrammetric data, obtained with telescopic lenses, is excellent for high-resolution planimetric mapping. However it can be employed only in a limited manner, involving sophisticated triangulation methods, for generating corresponding elevation data for topographic mapping and imaging, or for a

digital terrain model or database, unless supplemented by coincident altimeter readings [1, 7].

MGS is in a sun-synchronous polar orbit [6], so that an imaginary line from planet to sun always lies in (or makes a zero angle with) the plane of the MGS orbit. As a result, MGS always traverses (in a N-S direction) the planet's daytime face at midday at all (nadir) points of the surface. This enables photogrammetric data to be obtained at uniform lighting and shading angle. Since Mars rotates under the orbital plane both daily and annually, MGS will take a Martian year (about 2 Earth years) to complete an accurate survey of the entire planet. The initial polar orbit is highly elliptical, and a tight near-circular orbit, enabling the survey to begin, is expected in late 1998. Complete survey data should be available late in 2000. From this survey, detailed surface elevation, color, and some composition data will become available for the entire planet.

This means that it will shortly be possible to construct a very high resolution Mars Global Terrain Database (MGTDB). Such a database would not have been defensible earlier. The first extensive photography of the Martian surface was by the Mariner 9 spacecraft in 1971-72, which sent back over 7,000 images covering most of the planet. However the image quality had a resolution far too low for terrain database purposes; in addition elevation data was sparse. The situation was very much improved by the Viking Orbiters which arrived in 1976, and which during the next 4 years returned over 50,000 high quality images covering the entire planet [8, 9, 10]. There have been no further images from Mars until late 1997, when MGS sent back its first even better quality pictures.

Viking images cover large areas near the equator to a resolution of 7-30 meters, and at least 90% of the planet to a resolution of 100-150 meters. Accurate high-resolution terrain elevation data is lacking, however. A high-resolution global terrain database was not defensible with the Viking data, for three reasons: lack of uniform high resolution photogrammetric data for the whole planet, lack of uniform high resolution elevation data for the whole planet, and non existence of the powerful low-cost relational [11, 12, 13] and object-relational database techniques [14] and associated computer hardware resources that exist in the late 1990s. It is the expected high resolution altimeter, photogrammetric and geological data from MGS, coupled with the availability of low-cost powerful database techniques, that now make a high resolution MGTDB a feasible proposition for the next decade.

The primary purpose of an MGTDB is exploration of the planet. Because of the great distance of Mars, and oppositions only every two years, exploration of the planet even by robotic means is likely to be limited, expensive, and sporadic at best. Although robotic exploration will likely reveal detailed geological information at specific sites that could not be obtained from satellites, an effective and inexpensive exploration technique will involve using computer processing and a high resolution MGTDB [15].

Mars is a planet that generates intense scientific interest [16, 17]. Data returned from Mars by the early Mariner and Viking orbiters have revealed a densely featured planet that has undergone extensive cratering and great atmospheric, volcanic, tectonic and hydrological upheavals since coming into existence, in some ways similar to those of Earth during its early period [18, 19].

A properly designed MGTDB of sufficient resolution, supplemented with global geological data, would be a tool of incomparable value in research on the planet. And furthermore, once the data for it has been obtained, the cost of creating and operating the database, although not insignificant, would be negligible compared to the cost of even a single robotic expedition. Given the data for the database, current hardware costs for the database are of the order of M$0.5.

With the design presented in this paper, researchers would be able to retrieve the high resolution local terrain data, on scales ranging from the very large to the very small, for just about any combination of circumstances. Digital terrain data so retrieved would in most cases require considerable additional computer processing. Users would frequently want to have the grid of retrieved elevation and other data converted to an image of some kind, sometimes a map but more likely, for research purposes, a perspective or vista. Retrieved data could be used for automated generation of a variety of planimetric maps [20, 9], using cartographic projection techniques [7, 21], as well as images [22] and perspectives or vistas [23] for any position and direction. Many techniques developed for map and image generation using Earth digital terrain databases and GIS [1, 24, 25], such as ARCINFO [26], should be applicable to the terrain data extracted from the database. Fractal [27, 28, 9] and rendering [29, 30] techniques can be used to enhance images where data is insufficient.

Given these possibilities, it seems reasonable to investigate a possible structure design for a MGTDB. The design proposed is essentially entity-relationship [31], but it is presented as relations [11, 13], since it is expected to constructed as a relational database. However, with minor modification the design could be presented as an object-oriented database [14], although the author could find no advantage in using an object-oriented approach to the design problem. Functional, multivalued and join dependency considerations [11, 13] are largely irrelevant for a Mars database, since, apart from error correction, it would never be change updated. Nevertheless the design presented does avoid them. The suggested design is shown in Figure 1. Note that in this paper all sample queries to illustrate this database are in SQL [32, 11] although use of other powerful declarative languages is a possibility [33].

Geodetic considerations

Before considering an overview of the database design, a few geodetic matters [34] relating to Mars need to be considered. On Earth elevation data is with respect to a reference level or reference datum, which is normally sea level, reflecting an isogravimetric [35, 34]. With no sea on Mars, an equivalent reference datum must be derived on the basis of gravity and atmospheric pressure isobars. A complication is that Mars is even more spheroidal than the Earth. While Earth is a two-axial ellipsoid, Mars is best modeled by a 3-axial ellipsoid.

Using Viking data, in the 1980s the U.S. Geological Survey generated a Mars Atlas, updated since [9], of 140 quite detailed topographical maps at a scale 1:2000000, with kilometric contour lines. The elevations were with respect to a Mars reference datum established both from the planet's gravitational field, and from a reference corresponding to an atmospheric pressure of 6.1 millibars. This USGS Mars reference datum is a triaxial

ellipsoid with two semimajor axes of 3394.6 and 3393.3 km and a semiminor axis of 3376.3 km. This Mars reference datum appears to be a good standard, so that in the proposed database, elevation data will be with respect to this datum for a given latitude longitude coordinate. Note, however, that the triaxial ellipsoidal nature of Mars means that the exact length in meters of 1 second of arc, will vary slightly over the planet's surface and will need to be computed as required. A function (*trax()*) to acomplish this is included in the database as a virtual attribute. The zero of longitude on Mars is the center of a craterlet called Airy-0, at long 0.0, lat -0.5 [36].

Overview of the MGTDB design

The database will contain feature data and terrain data for a regular angular-coordinate (or arc raster) grid over the entire surface of Mars. At the core of the database is a 2000-Gigabyte relation called Terrain that contains the terrain data. It has a surface resolution everywhere, including at the poles, of 16.5 meters or better.

Data is recorded with respect to approximately 1-second of arc almost square grid elements. Each tuple of Terrain contains the latitude/longitude coordinates of a 1-degree curved rectangle, plus data for a N-S running column of 900 of these grid elements, a quarter degree of latitude long, called a N-S column grid, within the 1-degree rectangle. Thus for each one degree curved rectangle there are many Terrain tuples each denoting a N-S column grid. In the N-S direction there are four 900-grid-element N-S column grids per degree of latitude, and thus four tuples. The number of tuples in the E-W direction per degree of longitude, and thus the number of N-S column-grids, varies from 3600 per degree of longitude at the equator to 15 at the poles. Within any N-S column-grid each of the 900 grid elements has a N-S side length of exactly 1 second of arc, and the eastern and western side of a N-S column grid are meridians and intersect the poles. Also within any N-S column grid, the E-W side length of each grid element will normally vary from 1.0 down to 0.98 seconds of arc for latitudes less than 78, but from 1.0 down to 0 for the N-S column grid running from latitude 89.75 to 90. This arrangement ensures an almost square grid element, at least to within 2% for latitudes less than 78. Since 1 second of arc is close to 16.5 meters, it also ensures the resolution of 16.5 meters or better over the entire planet. (A N-S column grid can easily be visualized as equivalent to a stretch of highway in a N-S direction just under 15 km long (or 0.25 latitude degrees) and about 16.5 meters wide, with sides that are meridians, converging slightly in the polewards direction, but completely convergent in the case of a N-S column grid that ends at a pole.)

This Terrain relation design solves four practical problems associated with a relation for Mars global data. First it enables tuple size to lie well within disk track capacity with currrent technology, and will not lead to complications with blocking the records of the underlying Terrain file. Increasing resolution to 0.5 second (or about 8 meters) will not affect this compatibility. Second, the design accommodates the fact that as latitude increases the surface length of a degree of longitude decreases. It does so by using an almost fixed angular-coordinate grid element whose N-S sides are meridians and whose size stays almost constant in terms of surface length over the globe, but is sufficiently flexible to handle the convergence effect at the poles. The third problem solved is the geodetic problem of how to relate surface length of 1 second of arc to latitude,

longitude and elevation for a triaxial ellipsoid in a convenient manner; it is solved by placing the virtual attribute function *trax()* in Terrain.

The remainder of the database has to do with relating Martian feature-type relations to Terrain (Figure 1). The fact that Terrain is structured so that there is a set of tuples per 1-degree latitude-longitude rectangle, the number in the set falling with increasing latitude, is essentially transparent to users writing SQL expressions to retrieve Terrain data on the basis of specific features, even though there are many different types of features, each requiring a distinct relation. This is the fourth problem solved by the design. At least two of the feature-type relations needed, namely Crater and Valley, are involved in recursive relationships, recursive many-to-many in the case of Crater, and recursive one-to-many in the case of Valley.

Since all features have some attributes in common, each feature-type relations is in a 1:1 ISA relationship with a relation Feature that contains attributes common to all features. It is Feature that is linked to Terrain in a many-to-many relationship via a simple relationship relation FT. FT is estimated to be of the order of 1Mbyte in size, since it is likely to have a number of tuples comparable to the number of 1-degree rectangles on a sphere. The number of tuples in Feature is equal to the number of distinct feature instances named on Mars, e.g. Ares Valles, Pavonis Mons, etc., and is likely to grow with time. As a result of this design there are no feature-type relationship-attributes [37] in Terrain, so that it is possible to add additional feature-type relations in a modular manner. Since Feature-type relations are each in a 1:1 ISA relationship with Feature, a feature-type tuple inherits all the properties of the Feature supertype, including relationship participation.

The Feature/FT relations can be viewed as a "relationship bus" from Terrain, to which it is possible to attach any additional feature-type relation, regardless of any additional complexity due to recursive relationship participation. The design presented assumes that an initial implementation will include relations for only the most common features on Mars, namely the recursive Crater and Valley feature-type relations, and the ordinary Mountain feature-type relation.

From a data retrieval viewpoint, the structure seems to result in relatively straight-forward SQL expressions for terrain data retrieval. Where an SQL expression is complex, the complexity will not be due to unnecessary complexity in the database, but only to the complexity of the request, for example, a retrieval request involving one feature type within another feature of a different type, such as valleys in mountains, or one feature type within another of the same type, such as craters on top of craters - the recursive case. In the design, measured storage space is deliberately expended in order to reduce retrieval complexity. The total database size will be less than a percent greater than the size of the core relation Terrain, so that consuming storage space for non-Terrain relations to reduce complexity makes sense.

It should be understood at the outset that the design presented for a Mars Terrain Database must necessarily be different from the database designs with GIS for Earth use. A GIS product is essentially a component of a decision support system, for answering factual and analyical queries about what it is possible, e.g. where to site a pipeline or a bridge or a school [1]. A Mars terrain database has no such possible use in the foreseeable future (except perhaps on rare occasions to look for a landing site for a future Mars probe). Its sole purpose is exploration of the planet both on a large and small scale. It is

this goal that has determined the design presented here. Another difference is that databases for GIS for earth use, such as those used with such popular systems as ARC/INFO [26] are normally designed as thematic layers [1], which are integrated to satisfy queries [1, 38], the underlying layer being the natural terrain, and each additional layer involving something man-made. For example, if layer 0 is terrain, layer 1 could be street layout, layer 2 could be parks and layer 3 could be buildings, each layer using an underlying grid or raster approach for the data involved. With the Mars Terrain database there is no data about man-made entities. There is only data for the untouched natural terrain, or a single layer, so that the thematic layer approach of popular GIS software products, with their capability of handling a plethoria of man-made geographic objects, is largely irrelevant. Indeed, the SEDRIS data model for environmental databases, sponsored largely by DOD, is specifically aimed at synthetic environments and simulation (and related military operations). Popular GIS products do have some potential for use with the Mars Terrain database, however. GIS products are excellent for map generation, so that retrieved MGTDB Terrain data can be downloaded to a GIS product for generating maps.

2. The terrain data relation Terrain

The core of the database is the relation Terrain holding the terrain data for the planet. The basic data consists of terrain elevation, together with terrain color and composition data, versus latitude and longitude for the entire planet. A raster approach is needed and data is spaced on a grid of approximately square, approximately equal elements. Since data can be expected from MGS to a resolution of the order of 10 meters of surface, and since 1 second of arc is close to 16.5 meters on average, it is clear that the grid elements should be of the order of 1 second of arc, or even a fraction of a second. At the time of writing, the final orbit of MGS, and thus the data resolution, remains to be determined. In this paper we therefore assume what is likely to be a worst case for the grid, namely 1 second of arc. Changing it to 0.5 second, and a resolution of 8.25 meters, for example, will have no effect on the functionality of the database, apart increasing the required gigabytes by a factor of four, from 2,000 to 8000.

It should be obvious that given the enormous size of Terrain, a Terrain relation of the form:

Terrain (lat, long, x, y, elev, color, comp)

with one Terrain tuple per 1-second-of-arc square grid element, where x and y identify the position of a grid element within a 1-degree latitude-longitude rectangle, would involve an unnecessarily large number of Terrain tuples and waste of disk space. Admittedly, this version of Terrain appears to be similar to the Digital Terrain Elevation Data (DTED) files for Earth from the U.S. National Imagery and Mapping Agency (NIMA, formerly Defense Mapping Agency), except that those files that have been released (resolution levels 1 and 0) do not have resolution to anywhere near 1 second of arc; some work on a high resolution DTED files has begun however [39]. Also each DTED file is intended for military purposes for a specific area of the world, and not for a single global Earth Terrain Database.

A much more efficient database relation design, quite different from the DTED file design, which would very much reduce access times and access frequencies for Terrain, and to a considerable extent reduce underlying storage space, would be to store a North-South column of grid elements (referred to as an *N-S column-grid*) in a single Terrain tuple, as a 1-dimensional column-grid array. Such a N-S column-grid array consists of the data for exactly 900 approximately 1-second-of-arc square grid elements, each array element containing terrain elevation, color and composition data. Going from South to North by 1 degree (3,600 seconds) of latitude anywhere, within a 1 second E-W swath, gives rise to data for four Terrain tuples, each corresponding to a N-S column-grid of 900 grid elements, so that an N-S column grid is everywhere 0.25 latitude degrees long. Each of these four N-S column grids is identified in Terrain by a segment (*seg* value) integer running from 1 (polewards) to 4. The number of N-S column grids, and thus tuples, in the E-W direction per degree of longitude, is 3600Cos(lat) and thus falls off from 3600 per degree of longitude at the equator to 15 at latitude 89.75 at the poles. Allowance for this fall-off is called the arc raster approach to grid management [6]. In the E-W direction, the N-S column grids within a degree of longitude are identified by a *colgrid* integer running from 1 (westwards) to a value equal to 3600Cos(lat). The N-S column-grid within a 1-degree rectangle to which a tuple corresponds is thus determined by a *seg* and *colgrid* number in the tuple.

Within any N-S column-grid each of the 900 grid elements has a N-S side length of exactly 1 second of arc, with the eastern and western sides of the N-S column grid being meridians. Also within any N-S column grid, the E-W side length of grid element number 1 (towards equator) will be negligibly close to 1-second of arc; but as we proceed polewards along the N-S column grid to grid element 900, the E-W grid element side lengths gradually fall off; the fall-off averages about 1% overall, but rises to 2% at latitude 78; for any N-S column grid beginning at latitude 78, the E-W side lengths will be 1.0 second of arc for the element 1, falling to 0.98 second of arc for the grid element 900 towards the pole. For a final N.S column grid begining at latitude 87.75 and ending at the pole, the final grid element (no. 900) at the pole has an E-W length of 0, whereas the grid element at the other end (no. 1) has an E-W length close to 1.0 seconds of arc. This arrangement ensures an almost square grid element of almost constant side length in meters, at least to within 2% for latitudes less than 78. Since 1 second of arc is close to 16.5 meters, it also ensures the resolution of 16.5 meters or better over the entire planet.

Use of a single tuple to hold a terrain data for a N-S running column-grid of 900 approximately 1-second-of-arc grid elements is merely a somewhat original variation of the arc raster approach. With this variant, any given 1-degree latitude-longitude curved rectangle will be covered by a latitude-varying set of 4 x 3600Cos(lat) Terrain tuples for an average of 8892 over the planet.

This means that Terrain will have about 8,892 x 360 x 180, or 576 million tuples. Given a minimum of 4 bytes per grid element of a Terrain tuple, there will be 4 x 900 x 576 million bytes, or about 2000 gigabytes, in the relation Terrain.

Recall that for each degree of latitude, 4 distinct 900-grid-element tuples are used. Instead of 4 tuples, if a single 3600-grid-element tuple per degree of latitude were used, this would not do as well. The number of parallel N-S column-grid (and thus tuples) per degree of longitude is 3600Cos(lat), and falls from 3600 at the equator to 15 at the poles,

as discussed already. This implies an average falloff of about 4% tuples per degree of latitude, or 1% per quarter degree. Alernatively this can be viewed as a fall off of 0 tuples per quarter degree at the equator to 15 per degree (or a rate of 100%) at the poles. Thus there should be at least 4 tuples per degree of latitude in the database in order for the number of tuples per degree of longitude to fall by only 1% on average, for each quarter degree change in latitude. In order to have this gradual fall-off on average and yet a manageable maximum fall off at polar latitudes, and yet have no gaps in the coverage of the planet by the grid elements, and no deterioration in resolution anywhere, even at the poles, the E-W length of a grid element will normally vary slightly from 1-second to a few percent below 1-second of arc (but to 0 at the poles) within a N-S column-grid, as described above.

Since we are dealing with a triaxial ellipsoid, and elevations of the order of 10 kilometers above and below the reference datum, the exact surface length of 1-second of arc will vary, from place to place, by as much as 1%. The function *trax(lat, long, seg, colgrid, elev1)* will return the surface length of 1 second of arc at the begining (end towards equator) of an N-S column grid, using the elevation of the first grid element as a parameter. Thus *trax(56, 42, 3, 650, -3800)* would return the surface length of 1 second of arc at the beginning of the 650th N-S column grid west of longitude 42 and stretching between latitude 56.5 and 56.75, assuming the first grid element is 3.8km below the reference datum. The length of 1 second of arc at the other end of the N-S column grid is obtainable from data for beginning of the next tuple, that is, *trax(56, 42, 4, 650, -2400)*, assuming the first grid element of this tuple is now 2.4km below reference datum. It is important that *trax()* be sufficiently fine-grained. Mars has many cliffs that are kilometers high, dwarfing those of the Grand Canyon, so that there will be many cases of N-S column-grids straddling such cliffs, with a consequential tiny but abrupt change in the length of 1 second of arc and thus in grid sizes derived from this measure (remember, however, that the N-S sides of a N-S column grid are continuous meridians). The function *trax()* can be stored with the Terrain relation as a virtual attribute (i.e. *trax()* takes up no storage space in a tuple, but can be used like a conventional attribute name within SQL expressions with systems that allow this capability [40]. It takes its value from the value of the parameters, which are all real attributes in the tuple).

A tuple of a Terrain relation of this design has the structure:

Terrain (lat long, seg colgrid, elev1 col1 comp1, elev2 col2 comp2, ..., ...,
elev900 col900 comp900, trax(lat long seg colgrid elev1))

lat	short integer	latitude of S.E. corner of 1-degree latitude longitude rectangle
long	short integer	longitude of S.E. corner of 1-degree latitude longitude rectangle
seg	short integer	(1 to 4 polewards) in 1-degree latitude-longitude rectangle
colgrid	short integer	(1-3600 westwards) in 1-degree latitude-longitude rectangle
elev	integer	elevation data in tens of meters for the nth (1st to 900th in direction of pole) 1-second-of arc grid element in tuple
coln	short integer	color data for nth grid element in tuple
compn	short integer	surface composition data for nth grid element in tuple

trax() real surface length of 1 second of arc at start of N-S column grid parameters: *lat long seg colgrid elev1* as in Terrain tuple,
primary key: *lat long seg colgrid* composite
Indexes: secondary key index on *lat long* composite

The lat-long values in any Terrain tuple in the northern hemisphere are integer values for the S.E. corner of a 1-degree curved rectangle, so that lat-long data of 43, 215 means the curved rectangle between latitudes 43 and 44 North and between longitudes 215 and 216. Latitude degrees North are considered positive and degrees South negative. The primary key for the Terrain relation is the *(lat long seg colgrid)* composite. The Terrain relation would not be indexed in storage on the primary key, however; instead a secondary key index for the composite attribute *(lat long)* would be used, since data would almost always be retrieved on the basis of integer latitude longitude values, that is, for a 1-degree curved rectangle. Given the size of Terrain, some index optimization may be required in practice [41].

The terrain data for this rectangle, that is, the data for 4 x 900 x max(colgrid) grid elements, or for 4 x max(colgrid) tuples, can thus be simply retrieved by:

```
Select * from Terrain
where lat = 43 and long = 215
```

However, if data is desired for only a portion of a degree rectangle, this can easily be retrieved by specifying appropriate seg and colgrid values.

```
Select * from Terrain
where lat = 43 and long = 215
  and seg = 4 and colgrid >2000 and colgrid <= 3000)
```

This would retrieve exactly 1000 tuples, or data for 900,000 grid elements, covering part of the N.W. quadrant of the 1-degree rectangle.

An advantage of this structure is that any curved rectangle of data, from the vary large to the very small, can be retrieved just as easily. For example, suppose we want the rectangle of data for the great 3000-mile long Martian rift valley Valles Marineris in the S.W. quadrant. This valley stretches from about latitude 0 to 18 degrees South, and from about longitude 40 to 100 degrees West, with many tributary valleys. We would code:

```
Select * from Terrain
where lat <= 0 and lat > -18 and long <= 90 and long > 40
```

This would retrieve 18 x 60 sets of tuples from Terrain, each set containing a one-degree rectangle of data, corresponding to about 3.5 million square kilometers of terrain.

Overall, as already pointed out in the overview, this Terrain relation design solves four practical problems associated with a relation for Mars global data. The first problem solved concerns tuple size. In more, detail, assuming 4 bytes of data per grid element, we need close to 3600 bytes per tuple, which is well within disk track capacity with current

technology, and will not lead to complications with blocking the records of the underlying Terrain file. Increasing resolution to 0.5 second (or about 8 meters) will not affect this compatibility. With other designs that would much reduce the number of tuples in Terrain, the underlying records are simply too large for hardware compatibility. The reader is referred to the overview for the other problems solved. The design is highly flexible in that it enables users to extract data (a) on the basis of features and feature properties from the very large to the very small, as will be apparent later, and (b) on the basis of any curved latitude-longitude rectangle large or small, or even any portion of a 1-degree latitude-longitude rectangle all the way down to 1-second of arc.

3. Relating Terrain data to terrain features.

Humans usually do not think of locations in terms of angular coordinates, but in terms of feature types and feature names. Thus any generally useful database must relate features to the survey data in Terrain. Mars has a large number of different feature types, each with many instances, and the enormous variety of features easily exceeds that on Earth. Given that each type of feature will have distinct characteristics, a distinct relation will be needed for each feature type, and each of these relations must relate to Terrain, although not necessarily directly. The relationship between the planet's features and Terrain is therefore complex, especially when it is considered that features frequently overlap on the surface, for example, one crater partly obliterating two craters in a tributary valley of a main valley. These problems have been solved in the design in Figure 1. Before discussing that design further, we give a summary of the kinds of features occurring on Mars [36].

Main feature types on Mars

The International Astronomical Union, the body responsible for names on Mars, has adopted the following feature type names, using Latin. The list is not exhaustive.

Catena A chain of craters, not overlapping. An example is Ganges Catena in the SW quadrant [lat(-2,-3), long(70,65)]

Chasma A canyon, a depression with steep sides. An example is Coprates Chasma in the SW quadrant [lat(-10,-15), long(70,55)]

Crater Circular structure with a rim caused by a meteor impact. A crater may partly obliterate one or more earlier craters. Common on Mars.

Fossa A ditch-like, long, narrow, valley; can be either curved or straight. An example is Sirenum Fossa in the SW quandrant [lat (-35,-25), long(165,135)].

Mensa A table-like structure, with a flat top and steep sides. An example in the NE quadrant is Deuteronilus Mensa [Lat (45,40), long (345,340].

Mons A mountain, which can be of volcanic origin. Best known example is Olympus Mons in the NW quadrant, an extinct 15-mile high volcano [Lat 18, long 133]

Planitia A basin, or low-lying, smooth plain. Best known example is Chryse Planitia in the NW quadrant (the 'Plain of Gold', where Viking 1 landed in 1976) [lat (25,15),long (50,40)].

Rupes A steep cliff or scarp. An example is Amanthes Rupes which is up to 3 km high and stretches for 250 km.

Tholus An isolated hill, or dome-like mountain. An example in the NW quadrant is Jovis Tholus [lat 18, long 117].

Vallis A valley, or river-like winding channel, that may have tributary channels. Best-known example is Valles Marineris (Valleys of Mariner) in the SW quadrant, mentioned earlier. Also Ares Vallis where Pathfinder landed in 1997.

The database design in Figure 1 allows for addition of new feature types as required. Additional feature-type relations can be added without affecting the existing design (apart from extensions of the Feature and FT relations) Thus in this way the design is modularized. The relation for a feature type may be *ordinary* or *complex*. It is ordinary if a single relation can be used for feature instances with no recursive relationships, and complex if it is involved in a recursive relationship [31, 37].

A Crater relation is complex, since it must participate in a many-to-many (n:m) recursive relationship. A (top) crater may have impacted on one or more earlier (bottom) craters, and a (bottom) crater may have been impacted by one or more (top) craters; hence the n:m recursive relationship. A Valley relation is also complex, since it participates in a one-to-many (1:n) recursive relationship. A valley may be a tributary of another valley, and a valley may have one or more tributaries; hence the 1:n recursive relationship. A Mountain is an ordinary relation, since it does not participate in recursive relationships.

Scope of the current MGTDB design

The database structure shown in Figure 1 presents a design for an initial implementation. The three most common features on Mars are craters, mountains and valleys, so that to be useful the initial implementation would have to accommodate these. Note that not all craters, mountains and valleys on Mars have been named, so that the content of the relations for these features could be expected to increase over time. Since relations for other feature types are quite ordinary, like mountain, it is a fairly trivial matter for an implementor to expand the design presented and include relations for additional features. relations for crater, valley and mountain feature types are presented in Figure 1.

Feature sizes versus the 1-second-of-arc surface resolution

As will be seen presently, the database design allows for easy retrieval of the data

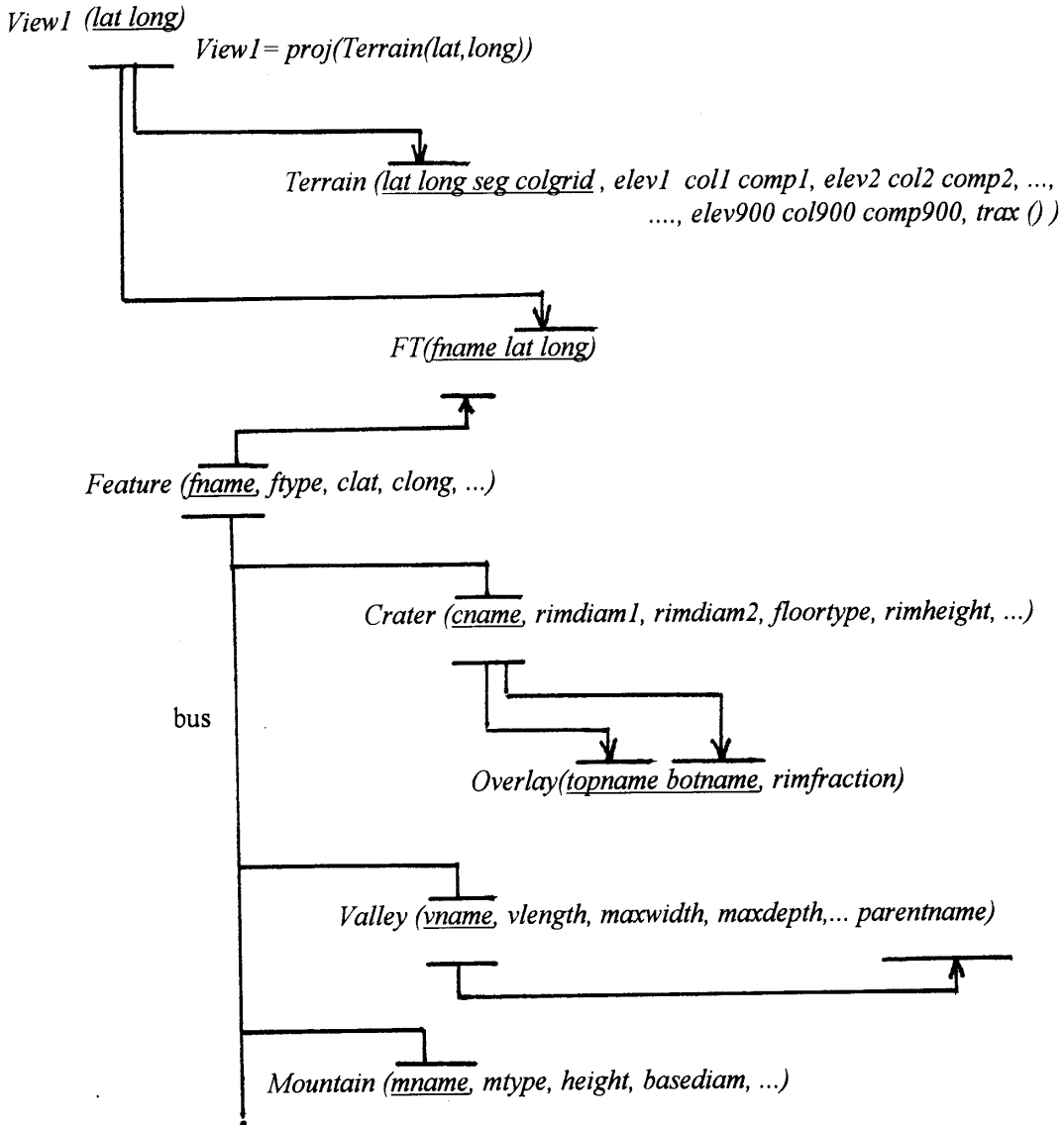

Figure 1 MGTDB structure. Connecting arrows show 1:n relationships. Connecting lines show 1:1 relationships. Primary keys are underscored. The derived view relation View1 is not stored but is included to better illustrate the relationships.

for very large features, even as large as the continental U.S., while at the same time it has a surface resolution down to 1-second-of-arc. Readers familiar with GIS may have difficulty grasping the purpose behind facilities apparently so far apart, until it is considered that we are not dealing with Earth but with an entirely new unexplored planet containing only natural features, ranging from vary large features to tiny features as yet unobserved and undocumented and perhaps only a few hundered meters in size, and most

importantly, where these smaller features can be observed from a variety of perpectives only via the data base - it will not be possible, as it is on Earth, to send a field worker out with a camera for a better look. Thus it has to be possible for a reseacher to use SQL to retrieve data for a feature on the basis of some attributes and/or relationships, and then explore subfeatures of that feature, perhaps hitherto unobserved, and subfeatures of that subfeature, all the way down to the limits of resolution; and furthermore any such subfeature, and subfeatures of that subfeature, must be observable from a wide variety of perspectives, even from the vantage point of a virtual walk-around or fly-around of the subfeature. The database design presented permits convenient retrieval of the data to enable such things to be accomplished via SQL retrievals, although generation of a variety of perspectives of subfeatures will require the application of vista and perspective generation software to the retrieved data. This can be done only (a) if the data base design allows for very high surface resolution, (b) if it has the structure to permit retrieval via attributes and relationships of features listed in the database, and (c) if it has the structure to allow addition of the new features as discovered. The design presented is in compliance with these requirements.

The Feature relation

All features, regardless of type, have common characteristics, and this is used to construct the Feature relation, with a tuple for every feature instance, regardless of type, in the database:

FT(fname, lat, long)
Feature (fname, ftype, clat, clong, ...)

FT:	*fname*	string	feature name
	lat	short integer	latitude
	long	short integer	longitude
	primary key:	*fname lat long*	composite
	index:	primary key index	

Relationships: *FT* relates to *Terrain* via the *lat long* composite
FT relates to *Feature* via *fname*

Feature:	*fname*	string	feature name
	ftype	string	feature type, e.g. crater, valley
	clat(deg min sec)	integer integer integer	latitude center of feature
	clong (deg min sec)	integer integer integer	longitude center of feature
	...		
primary key:	fname		
indexes:	primary key index on *fname*		
	secondary key index on *ftype*		

There will be in a 1:1 ISA relationship [31, 12, 21] between Feature and each of the specific feature-type relations, such as relations Crater, Valley and Mountain. Thus a

tuple in feature-type relation Mountain for *mname* = *'Olympus Mons'* inherits the properties of the corresponding tuple in Feature for which *fname* = *'Olympus Mons'*.

The important design consideration concerns relating the quite different relations, Crater, Valley, Mountain, and so on, to the relation Terrain. This problem is solved by enabling the essentially many-to-many (n:m) relationship that must exist between Feature and Terrain. [Actually the n:m relationship is between Feature and View1, as shown in Figure 1; View1 simply lists all the degree rectangles for the planet; for each degree rectangle there can be many features and for each feature many degree rectangles. View1, obtained from a projection of attributes *lat long* of Terrain, is not part of the database and is not stored, but is useful for better understanding the relationships.] The enabling relation is FT, and its single but crucial function is to relate a tuple of Feature (representing a feature-type instance) to the tuples in Terrain that cover the terrain encompassed by the feature. Remember, however, that there is a set of tuples of Terrain for each 1-degree rectangle of terrain data. The relationship between Feature and Terrain is an essentially many-to-many because a specific 1-degree rectangle of terrain data, a set of tuples of Terrain, can be terrain in more than one feature (e.g. the crater in a valley, or the crater on top of another crater), and vice versa.

An example will clarify this. Suppose a fictional crater called C3 in the side of a fictional mountain called M6. Suppose the mountain is centered in the curved square between lat (50, 46) and long (104,100) with peak (center) at about (48,18; 102,12). Suppose the crater lies inside the square with lat (48, 46) and long (102, 100) with center at (47,06; 101,12). The relevant tuples in the Feature, FT and Terrain relations would be:

fname	***ftype***	***clat***	***clong***
...	...	...	...
C3	crater	47 06 00	101 12 00
...	...	...	...
M6	mountain	4818 00	102 18 00
...	...	...	...

Feature

fname	***lat***	***long***
...	...	...
C3	47	100
C3	47	101
C3	46	100
C3	46	101
M6	49	103
...	...	...
M6	48	100
M6	47	103
M6	47	102
M6	47	101
M6	47	100
M6	46	103
M6	46	102
M6	46	101

Lat	***long***	***seg colgrid elev1 col1 comp1*** ...
...	...	... ...
49	103	
...	...	
48	100	
...	...	
47	103	
...	...	
47	102	
...	...	
47	101	
...	...	

M6	46	100		47	100
...	...	...		...	...
		FT		46	103
				...	...
				46	102
				...	...
				46	101
				...	...
				46	100
					Terrain

From this it is clear that for one Feature tuple, for example for crater C3, there are many (4) degree-rectangle tuple sets in relation Terrain that contain the terrain data for C3. Conversely, for a 1-degree-rectangle Terrain tuple set, for example the set of tuples each with (lat long) value (47 101), there are many (2) tuples in Feature, one involving crater C3 and one involving mountain M6. This structure involving relations Feature, FT and Terrain enables survey and other data to be retrieved on the basis of a specific feature:

R1: Get terrain data for the mountain Pavonis Mons.

```
Select * from Terrain
where lat long in (select lat long from FT
            where fname = 'Pavonis Mons')
```

R2: Get terrain data and mountain name for every mountain in the NW quadrant south of latitude 20.

```
Select FT.fname, Terrain.*  from FT, Terrain
where FT.lat = Terrain.lat and FT.long = Terrain.long
   and FT.fname in (select fname from Feature
            where ftype = 'mountain'
            and clat.deg < 20 and clat.deg  > 0
            and clong .deg > 0 and clong.deg < 180)
```

R3: Get the name of each valley in which there is at least one embedded crater.

```
Select XF.fname from Feature as XF
where XF.ftype = 'valley' and XF.fname in
   (select XFT.fname from FT as XFT
    where where XFT.lat XFT.long in
      (select YFT.lat YFT.long from FT as YFT
       where YFT.fname in
         (Select YF.fname from Feature as YF
          where YF.ftype = 'crater'))
```

Notice that this information can be retrieved without using Terrain.

Alternative to the Feature/FT relation approach

A criticism of the use of the relations Feature and FT, as described above, is that the data in FT could be computed if coordinates for a rectangle enclosing the feature were stored in Feature, instead of the location of the center. It could therefore be argued that we should replace both Feature and FT by a single alternative relation FTFeature:

FTFeature (fname, ftype, maxlat, minlat, maxlong, minlong)

where fname is the primary key, ftype gives the type of feature, e.g. crater, or valley, and maxlat, minlat, maxlong and minlong give the latitudes and longitudes for the curved rectangle that enclosed the feature. The former clat in Feature can now be computed from *(maxlat - minlat)/2* and clong from *(maxlong - minlong)/2* to give the latitude and longitude of the center point of the feature.

We can admittedly use maxlat, minlat, maxlong, and minlong directly to relate FTFeature to Terrain, but only at the expense of either more complex or more tedious retrieval expressions. The reader who tries the rewrite of R1, R2 and R3 above with FTfeature will be quickly convinced that this is not a good alternative. The problem gets worse when we bring feature-type relations like Valley, Mountain and Crater into the retrieval expressions

An alternative to this complexity and tedium would be to construct two views [11, 12] based on FTFeature, one equivalent to FT and the other equivalent to Feature. However this would be at the undesirable expense of very much increased computing time for retrievals.

An FT tuple is unlikely to exceed 20 bytes, so that the maximum size of FT is of the order of 1 Mbyte (derived by assuming that each of the 360 x 180 degree rectangles has an entry in FT), which is negligible compared with the size of Terrain, estimated at 2,000 Gigabytes. Since the cost of data storage resources is falling, this author's preference is to add the resource of FT to the database and gain the reduction in complexity. Some readers may not agree, but we shall assume the Feature and FT relations design as in Figure 1. From a systems perspective, this is just another example of the trade-off between resources invested versus complexity of current operations.

4. The feature-type relations

Although the availability of the Feature relation imparts considerable flexibility, in practice users will want to get at Terrain and other data on the basis of characteristics of features, such as diameter of a crater or height of a mountain, etc. For this we need the feature-type relations, such as Crater, Mountain, Valley, and their relationship to Feature, as shown in Figure 1.

Crater relation

Since craters are common on Mars and since they can overlay one another, any Crater relation has to be in a n:m relationship with itself, that is, in a recursive relationship.

In Figure 1, the relationship relation Overlay is used to enable the recursive relationship, that is, for craters we have:

Feature (fname, ftype, clat, clong ...)
Crater (cname, rimdiam1, rimdiam2, floortype, rimheight, ...)
Overlay(topname, botname, rimfraction)

Crater:	*cname*	string	crater name
	rimdiam1	real	outer rim diameter in km
	rimdiam2	real	inner rim diameter in km
	floortype	string	type of crater floor, e.g. sandy
	rim height	integer	mean rim height above floor
	...		

primary key: *cname*
indexes: index on primary key
relationships: relates to *Feature* and *FT* via *fname* and *cname*

Overlay:	*topname*	string	name of crater on top of crater named in botname
	botname	string	name of crater underneath crater named in topname
	rimfraction	real	fraction of botname crater rim destroyed by topname

primary key: *topname botname* composite
indexes: secondary key index on *topname*
secondary key index on *botname*
relationships: relates recursively n:m to *Crater* via *topname, botname* and *cname*
relates to *Feature* and *FT* via *topname* and *fname*
relates to *Feature* and *FT* via *botname* and *fname*

In the relation Crater, many other technical attributes about craters beyond the scope of this paper can also be included.

In the relation Overlay the primary key is the composite *topname botname*. The crater topname will have been fashioned on top of prior existing crater botname, such that a percentage of the rim (*rimfraction*) of the botname crater has been obliterated by the meteor impact forming topname. To see how Overlay enables the recursive relation, consider an area of crater congestion, as occurs in the SE quadrant, with fictional craters C1, C2, C3, C4 and C5. Craters C1 and C1 are about the same size, 50 km in outer diameter, and were formed first. They are close together; the rims are about 30 km apart but do not overlap. Then a larger crater C3, about 75 km in outer diameter is formed partly on top of both C1 and C2, so that a fraction 0.25 of the rim of each of C1 and C1 is destroyed. Then a further meteor impact, partly on top of C3 and C2, forms crater C4. C4 is 60 km in outer diameter and destroys a fraction 0.3 of the rim of C3 and a further fraction 0.2 of the rim of C2. Finally a small crater C5, 20 km in outer diameter, is formed in the middle of the floor of C4, without destroying any of the C4 rim. The database will show:

cname rimdiam1 rimdiam2 ...

...	...	...
C1	...	50
C2	...	50
C3	...	75
C4	...	60
C5	...	20
...	...	...

Crater

topname	*botname*	*rimfraction*
...	...	...
C3	C1	0.25
C3	C2	0.25
C4	C2	0.20
C4	C3	0.30
C5	C4	0
...	...	...

Overlay

It should be apparent that crater C3 has two child craters C1 and C2, that crater C4 has two child craters, that C5 has one child crater, that C1 has one parent crater C3, that C2 has two parent craters C3 and C4, and that C4 has one parent crater C5. It is in this way that relation crater participates in a recursive many-to-many relationship. Some retrievals will illustrate further:

R4. For each case of a (child) crater with a single (parent) crater completely inside the child crater, with the rims of both parent and child fully intact, retrieve the parent and child crater names, and the relevant Terrain data.

```
select Overlay.topname, Overlay.botname, Terrain.*
  from Overlay, Terrain, FT
  where Overlay.botname = FT.fname
    and FT.lat = Terrain.lat and FT.long = Terrain.long
   and Overlay.rimfraction = 0
   and /* eliminate cases where rims not fully intact */
    not exists (select L1.* from Overlay as L1
       where (Overlay.botname = L1.botname and rimfraction > 0)
        or (Overlay.Topname = L1.botname and rimfraction > 0) )
```

R5. Get the crater name and terrain data for each crater in the Northern hemisphere with a sandy floor that has a partly destroyed rim.

```
Select FT.fname, Terrain.* from FT, Terrain
where FT.lat = Terrain.lat and FT.long = Terrain.long
 and FT.fname in (select fname from Feature
            where clat > 0 and fname in
    (select cname from Crater
     where floortype = 'sandy'
      and  cname in (select botname from Overlay
                where rimfraction > 0))
```

R6. Get latitude and lontitude data for the curved square that contains the crater Cassini.

```
Select max(lat), min(lat), max(long), min(long) from FT
```

```
where fname = 'Cassini'
```

This design allows for retrieval of terrain and other data for every conceivable crater configuration that occurs in any crater-congested region of Mars.

Valley relation

Valleys on Mars can have tributary valleys, so that a valley resembles a branch of a tree. Such a structure is easily handled by a recursive 1:n relationship. The Valley part of the database in Figure 1 is:

Feature (fname, ftype, clat, clong)
Valley (vname, vlength, maxwidth, maxdepth,... parentname)

Valley:	*vname*	string	name of valley
	vlength	real	length of valley in km
	maxwidth	real	maximum width of valley in km
	maxdepth	real	maximum depth of valley in km
	parentname	string	name of the valley of which this is a subsidiary valley

primary key: *vname*
indexes: index on primary key
relationships: relates recursively 1:n to itself via *vname* and *parentname*
relates to Feature and FT via *fname* and *vname*

In Valley, for a given tuple for valley V1, the attribute parentname gives the name of the valley for which V1 is a tributary valley. Suppose this parent is V7. Then valley V7 may have many tributary valleys, but at least one, namely V1. For a given valley it is clear that there can be only one parent. However for a given valley there can be many tributaries or child valleys. Thus Valley is in a recursive 1:n relationship. Some retrievals will illustrate:

R7. Get name, length, and maximum depth of each tributary valley in the Southern hemisphere that itself has more than 4 tributary valleys.

```
Select vname, vlength, maxdepth from Valley
where parentname is not null
  and vname in (select fname from Feature where clat < 0)
  and 4 < (select count (XValley.*) from Valley as XValley
        and XValley.parentname = Valley.vname)
```

R8. Get the valley name and terrain data for each valley in the Western hemisphere that has no tributary valleys.

```
Select Valley.vname, Terrain.* from Valley, Terrain, FT
 where Valley.vname = FT.fname
 and FT.lat = Terrain.lat and FT.long = Terrain.long
```

```
where vname in (select vname from Feature
        where clong < 180 and clong > 0)
and not exists (select XValley.* from Valley as XValley
        where XValley.parentname = Valley.vname)
```

The structure of Valley thus allows for navigating through a valley complex with many levels of tributaries. This would be important for a virtual exploration of a valley of the complexity of Valles Marineris, which not only has tributary valleys but overlaps many canyons (chasma).

Ordinary feature-type relations

Crater and Valley relations are significant for their additional participation in recursive relationships. Many ordinary feature-type relations can be included in the database design as well. The relation Mountain is included as shown in Figure 1:

Mountain (mname, mtype, height, basediam, ...)

has mname as primary key. The relation Mountain, together with its supertype Feature, would be used in the retrieval:

R9. Get the names, heights and terrain data for volcanoes lying on the equator.

```
Select Mountain.mname, Mountain.height, Terrain.
        from Mountain, Feature, FT, Terrain
  where Mountain.mname = Feature.fname
      and Feature.fname = FT.fname
   and FT.lat = Terrain.lat and FT.long = Terrain.long
   and Mountain.mtype = 'volcano'
   and Feature.clat.deg = 0.
```

Any additional feature-relation, such as a Plain relation, can be added to the database without altering the Terrain relation. It is necessary only to insert additional tuples into Feature, one for every tuple in Plain, and to insert additional tuples into FT, one for every degree rectangle of terrain taken up by each Plain instance. Thus Feature and FT can be looked upon as a bus for hanging feature-type relation modules onto, for linking to the fundamental Terrain relation.

5. Summary

This paper has presented a modularized design for a Mars Global Terrain Database to be based on data transmitted from a Mars survey being undertaken by the Mars Global Surveyor Spacecraft. The database will contain feature data and terrain data, including terrain elevation, for a regular grid over the esurface of Mars. Elevation is with respect to a triaxial ellipsoidal reference datum developed for Mars by USGS. At the core of the database is a 2000-Gigabytes relation called Terrain that contains the terrain data.

Data is recorded with respect to approximately square 1-second-of-arc grid elements. Each tuple of Terrain contains the latitude/longitude coordinates of a 1-degree curved rectangle, plus data for a column of 900 grid elements, called a N-S column grid, for a N-S running column of 900 1-second-of-arc grid elements within the 1-degree rectangle. The side of a N-S column grid are meridians. Each grid element is about 16.5 meters square (smaller near the poles), giving a surface resolution of 16.5 meters or better. The number of Terrain tuples required to cover a quarter-degree latitude one-degree longitude rectangle falls from 3600 at the equator to 15 at the poles. The triaxial ellipsoidal nature of Mars is allowed for by means of a virtual attribute geodetic function *trax()* in Terrain that returns the surface length of 1 second of arc at the beginning of each N-S column grid.

The remainder of the database has to do with relating Martian feature-type relations to Terrain. Since all features have some attributes in common, each feature-type relations is in a 1:1 ISA relation with a relation Feature that contains attributes common to all features. It is Feature that is linked to Terrain in a many-to-many relationship via a simple relationship relation FT. There are no feature-type relationship-attributes in Terrain, so that it is possible to add additional feature-type relations in a modular manner.

The Feature/FT relations can be viewed as a bus from Terrain, to which it is possible to attach any additional feature-type relation, regardless of any additional complexity due to recursive relationship participation. Three feature type relations, for the most common Martian features, are assumed for an initial implementation, namely the recursive Crater and Valley relations, and the ordinary relation Mountain.

From a data retrieval viewpoint, the structure results in relatively straight-forward SQL expressions for terrain data retrieval. Where an SQL expression is complex, the complexity will not be due to unnecessary complexity in the database, but only to the complexity of the request. In the design, measured storage space is deliberately expended in order to reduce retrieval complexity.

A much smaller prototype version of the database proposed in this paper can be contructed while awaiting MGS data. Such a prototype would be for a smaller mid-latitude section of the planet, perhaps in the interesting Valles Marineris region, about 5 x 5 degrees in size, in an area of a multiplicity of small features. The size of the database prototype would then be a reasonable 1.0 Gigabytes.

References

1. P.A. Burrough and R. McDonnell. Principles of Geographic Information Systems, Oxford University Press, Oxford, England, 1998.

2. M. Juppenlatz and X. Tian. Geographic Information Systems and Remote Sensing, McGraw-Hill, New York, 1996.

3. A. Showengerdt. Remote Sensing: Models and Methods for Image Processing, Academic Press, New York, 1997.

4. R. K. Vincent. Fundamentals of Geological and Environmental Remote Sensing, Prentice-Hall, Englewood Cliffs, NJ, 1997.

5. T.A. Mutch, R.E. Arvidson, J.W. Head, K. L. Jones and R.S. Saunders. The Geology of Mars, Princeton University Press, Princeton, New Jersey, 1976.

6. C. Elachi. Introduction to the Physics and Techniques of Remote Sensing, Wiley, New York, 1987.

7. D. H. Maling. Coordinate Systems and Map Projections, Philip & Son, London, 1973.

8. H.H. Kieffer, B.M. Jakosky, C.W. Snyder and M.S. Mathews, M.S. (Editors). Mars, University of Arizona Press, Tucson, 1992.

9. USGS. Atlas of Mars, Version 1.0.9, USGS Information Service, Denver Co., 1995.

10. Viking Lander Imaging Team, The Martian Landscape, NASA SP-425, Science & Technology Information Office, Washington, D.C, 1978.

11. C. J. Date. Database Systems, 6th Edition, Addison Wesley, Reading, Mass., 1995.

12 . C. J. Date and G. Darwen. A Guide to the SQL Standard, 3rd Edition, Addison Wesley, Reading, Mass, 1993.

13. A. Silberschatz, H.F. Korth and S. Sudarshan. 1997. Database System Concepts, 3rd Ed., McGraw-Hill, New York, 1997.

14. A. Dogac, M. T. Ozsu, A. Biliris, and T. Selis (Editors). Advances in Object-Oriented Database Systems, NATO ASI, Series F, Vol 130, Springer-Verlag, New York, 1994.

15. H. Samet, and W Aref. 'Spatial data models and query processing', in: Modern Database Systems, edited by W. Kim, ACM Press/Addison Wesley, pp338-360, 1995.

16. M. H. Carr. The Surface of Mars, Yale University Press, New Haven, Connecticut, 1984.

17. T. R. Watters. Planets, a Smithsonian Guide, MacMillan, USA, 1995.

18. M. H. Carr, et al. 'Martian impact craters and the emplacement of ejecta by surface flows', J. Geophys. Res., 82 , No. 4, pp35-64, 1997.

19. V.C. Gulick, and V.R. Baker. Origin and evolution of valleys on Martian volcanoes, J. Geophys Res., 95, No.14, pp235-344, 1990.

20. R. Greeley, and R. Bateson. NASA Atlas of the Solar System, Cambridge University Press., 1997.

21. A. K. Skidmore. Terrain position as mapped from a gridded digital elevation model, Int. J. Geographic Information Systems, Vol 4, 33-49, 1990.

22. J. D. Foley, A. van Dam, S.K. Feiner and J. F. Hughes. Computer Graphics, Principles and Practice, 2nd Ed., Addison-Wesley, Reading, Mass., 1990.

23. B. Fishman, and B. Schachter. 'Computer display of height fields', Computers and Graphics, Vol 5, pp53-60, 1980.

24. B. Falcidieno and C. Pienovi. 'A feature-based approach to terrain surface approximation', Proc. 4th Int. Symp. on Spatial Data Handling, Zurich, Switzerland, Vol 1, pp190-199, 1992.

25. L. Floriani and P. Gattorna. 'Spatial queries on a hierarchical terrain model', Proc. 6th Int. Symp. on Spatial Data Handling, Edinburgh, Scotland, Vol 2, pp819-834, 1994.

26. ARC/INFO, Environmental Systems Research Institute, 380 New York St., Redlands Calfornia.

27. R.G. Lathrup. 'Identifying structural self-similarity in mountainous landscapes', J. of Landscape Ecology, Vol 6., pp233-238, 1992.

28. D. Saupe. 'Algorithms for random fractals', in: The Science of Fractal Images, edited by H. Peitgen, Springer-Verlag, New York, 1988.

29. A. Fournier, D. Fussel, and L. Carpenter. 'Computer rendering of stochastic models', Communications of the ACM, Vol 25, pp371-384, 1982.

30. R. M. Voss. 'Fractals in nature, from characterization to simulation', in: The Science of Fractal Images, edited by H. Peitgen, Springer-Verlag, New York, 1988

31. C. Batini, S. Ceri and S.B. Navathe.1992. Database Design: An Entity-relationship Approach, Benjamin Cummings, Redwood City, Ca, 1992.

32. ANSI. Database language SQL, ANSI X3, 135-1992, American National Standards Institute, New York, 1992

33. J. Bradley. 'Extended relational algebra for reduction of natural quantifier COOL expressions', J. Systems Software, Vol 33, pp87-100, 1996.

34. James R. Smith. Introduction to Geodesy, Wiley, New York, 1997.

35. I. I. Mueller and R.H. Rapp. 'Horizontal and vertical geodetic datums', in: "Reference Frames in Astronomy and Geophysics", edited by J. Kovalevsky, I. I. Mueller and B. Kolaczek, Kluwer Academic Publishers, London, 1989.

36. P. Moore. Guide to Mars, Lutterworth Press, London, 1997.

37. J. Bradley. File and Database Techniques, Holt, Rinehart & Winston, New York, 1982

38. O. Kufoniyi, and M. Pilonk. 'A vector data model integrating multitheme and relief geoinformation' Proc 6th Int. Symp. on Spatial Data Handling, Edinburgh, Scotland, Vol 2, pp1061-1071 , 1994

39. DTED, National Imagery and Mapping Agency, Bethesda, Maryland.

40. M. Stonebraker, J. Anton and E. Hanson. 'Extending a database systems with procedures', ACM Trans. on Database Systems Vol 12, No. 3, pp350-376, 1987.

41. R. Ramesh, A. J. G. Babu and J. P. Kincaid. 'Index optimization: Theory and experimental results', ACM Transactions on Database Systems, Vol 14, No. 1, pp41-74, 1989.