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 grid elements over the surface of Mars. A 400 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 a uniform 16.5 meters everywhere with a 1-second grid. 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 are 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.

Key words database, elevation, grid, Mars, Mars Global Surveyor, relations, resolution, terrain, U.S. Geological Survey

1. Introduction

During the last two decades there has been increased research into Geographic Information Systems involving digital terrain databases [12, 13, 20, 29,], often obtained by remote sensing [5, 26, 33] of the surface of the earth. Earth-terrain databases have multiple uses, ranging from military to civil engineering, hydrological, commercial aircraft safety and prospecting applications. But now, with the successful arrival of the Mars Global Surveyor (MGS) spacecraft in Mars polar orbit in the fall of 1997, extensive new digital terrain database application possibilities are emerging, initially mainly in geological research, for the essentially unexplored terrain of the planet Mars. The MGS spacecraft should return the data necessary for a high resolution Mars Global Terrain Database. The spacecraft is equipped with a laser altimeter 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 [5]. However it can be employed only in a limited

manner, involving sophisticated triangulation methods, for generating corresponding elevation data for topographic mapping and imaging [22, 33], or for a digital terrain model or database, unless supplemented by coincident altimeter readings.

MGS is in a sun-synchronous polar orbit [11], so that an imaginary line from planet to sun always lies in (or makes a zero angle with) the plane of the MGS orbit. Sun-synchronous polar orbits have been in common use for two decades by satellites from many countries conducting Earth surveys (although the orbital plane is often a fixed non-zero degree angle with the line to the sun). 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. Although this is of little consequence for altimeter data capture, it enables photogrammetric data to be obtained at uniform lighting and shading angle. Since the planet rotates under the orbital plane both daily and annually, MGS will take a 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.

From this survey, detailed surface elevation, color, and some composition data will become available for the entire planet for the first time. Every area will be surveyed in both summer and winter. This is important because Viking lander photographs [35, 36] have shown that in Winter, even at latitudes as low as 22, the surface can be covered by light water-ice snow during the entire day, a phenomenon that can be observed from Earth. If no significant spaceprobe malfunction, and no dust storms on a global scale to delay the photographic data, complete survey data should be available in the year 2000,

This means that for the first time, 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, while more than sufficient to reveal the much of true nature of the Martian terrain for the first time, and arouse the interest of the scientific community, had a resolution far too low for terrain database purposes; in addition elevation data was sparse and essentially only estimates. 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. These Martian images intensified interest and have been much studied ever since [7, 19, 35, 36, 38]. There have been no further images from Mars until late 1997, when MGS sent back its first even better quality pictures.

The Viking Orbiters had orbits with modest equatorial inclination, and 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 [8, 9, 31] and object-oriented database techniques [10] and associated computer hardware resources that exist in the late 1990s. It is the expected high resolution altimeter [27], 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.

Such a database would have many uses, and currently it is impossible to foresee all of them. Because of the great distance of Mars (9 months with current technology), and oppositions only every two years (last opposition: March 1977; next: April 1999), exploration of the planet even by robotic means is likely to be limited, expensive, and sporadic at best. Although robotic exploration will reveal detailed geological information at specific sites that could not be obtained from satellites, a very effective and inexpensive exploration technique will involve using computer processing and a high resolution MGTDB (supplemented by geological data as it becomes available).

Mars is a planet that generates intense scientific interest [7, 19, 23, 25]. 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 [6, 7, 18] . The difference between the two planets in terms of surface terrain generation seems to be due to two factors: first, Mars had a crust that was too thick to allow for mobile tectonic plates as on Earth, but thin enough to cause large volcanoes of a size never seen on Earth, and generate widespread tectonic phenomena in the form of surface fractures, faults and scarps, that sometime even dislocate craters, as well as a rift valley system like Valles Marineris, which is on a scale far surpassing anything comparable on Earth; second, the effects of surface erosion due to atmospheric and hydrological forces was much less than on Earth, where, for example, the record of craters caused by early bombardment, and their interaction with tectonic forces, is all but gone [38]. The study of the unfolding of Mars is necessarily still in its infancy but because the geological record there is far better preserved than on Earth, it is a field of great and justified promise.

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. Researchers would be able to retrieve the high resolution local terrain data for any combination of circumstances, for example, for any location and for any feature on the planet, for features with specific characteristics, for areas (common on Mars) where different types of features overlap and interact, and for features with similar topological and geological properties similar to a specific feature under study. (Such retrievals are easily possible with the database design proposed in this paper). Digital terrain data so retrieved could be used in many ways, although in most cases considerable additional computer processing will be needed. Uses include automated generation of a variety of planimetric maps [5, 17, 22, 17], as well as images [16] and perspectives or vistas [15] for any position and direction, for a multiplicity of research purposes. Many techniques developed for map and image generation from Earth digital terrain databases [12, 32] should be transferable to the terrain data extracted from the database, for example, application of fractal techniques [14, 21, 30, 37]

Given these possibilities, it seems reasonable to investigate a possible structure for a MGTDB, and in this paper we discuss a design for one resulting from such an investigation. The design approach is essentially entity-relationship [2], but the design is presented as relations [8, 31] to enable sample SQL [1, 9] queries to be carried out. With

minor modification it could be presented as an object-oriented database [10]. Functional, multivalued and join dependency considerations [2] 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. A wealth of current information on the MGS project is available from NASA websites.

Geodetic considerations

Before beginning the discussion, a few geodetic matters relating to Mars need to be considered. Since elevation data is fundamental to any MGTDB, the question that immediately arises is: elevation with respect to what? On Earth elevation data is with respect to a reference level or reference datum, which is normally sea level, reflecting an isogravimetric [24]. 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. The Mars equator is slightly elliptical, whereas the Earth equator is a circle, and the Mars meridian also forms a distinct ellipse, as does that on Earth. Thus 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 [35], of 140 quite detailed topographical maps at a scale 1:2000000, with kilometric contour lines. The elevations were with respect to a reference datum established both from the planet's gravitational field, and from a reference corresponding to an atmospheric pressure of 6.1 millibars. This basic 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 UDGS-derived 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. This ellipsoidal datum gives rise to a further problem for the database designer, however. On a precisely smooth spherical planet a degree of latitude anywhere or longitude at the equator will always correspond to a fixed surface length (length of the arc), namely 3.14D/360. But on an ellipsoidal planet, particularly one where there are great surface elevation variations with respect to the reference datum, as is the case on Mars, the length on the surface corresponding to a degree will vary, and must be computed from the reference ellipsoid. Consequently, as a basic principle, terrain elevation and other data in a database relation should not relate directly to grids with a sides of fixed length in meters. Instead data in a database relation should be stored in relation to grids whose sides are constant in terms of angular coordinates. The exact side length in meters of such angular-coordinate grid elements will vary slightly over the planet surface but can be computed as required.

The zero of longitude on Mars is the center of a craterlet called Airy-0, called after Sir George Airy [23], the 19th century British Astronomer Royal who was largely responsible for making Greenwich the zero for longitude on Earth. Airy-0 is at long 0.0, lat -0.5.

****

Terrain (lat long seg, colgrid, elev1 col1 comp1, elev2 col2 comp2, ...,
...., elev600 col900 comp900, trax ())

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

Crater (cname, rimdiam1, rimdiam2, floortype, rimheight, ...)
Overlay(topname, botname, rimfraction)

Valley (vname, vlength, maxwidth, maxdepth,... parentname)

Mountain (mname, mtype, height, basediam, ...)
...
<other feature type relations>

Figure 1
****

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. Data is spaced on a grid of slightly curved, 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, 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. As will be clear shortly, 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.

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

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

with one Terrain tuple per grid element, and latitude and longitude data to one second of arc, would involve an unnecessarily large number of Terrain tuples (1.6 10 12) and waste of disk space.

A much more efficient design, which would very much reduce access times and access frequencies for Terrain, and to a considerable extent 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. This 1-dimensional N-S column-grid array consists of the data for exactly 900 1-second 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, with a 1 second longitudinal swath, gives rise to data for four Terrain tuples, each corresponding to a N-S column-grid of 900 grid elements. The set of four tuples concatenated would correspond to a N-S column of 3600 1-second grid elements. In addition, at the equator, in going East West by 1 longitude degree, there would be exactly 60X60 or 3600 parallel 900-element N-S column-grids, and hence 3600 tuples. Thus, at the equator, the data from a 1-degree square of surface would be in 4 X 3600 or 14,400 900-grid-element Terrain tuples.

Four tuples span a degree of latitude at all latitudes. The situation is different with longitude. On a sphere, the surface length of a degree of longitude falls from 3.14D/360 at the equator to 0 at the poles, so that the number of parallel 900-element column-grids per degree of longitude will fall from 3600 at the equator to zero at the poles (and 1 for latitude 89.75). However the falloff is not linear with respect to latitude, but linear with respect to Cos(lat), since the length of a degree of longitude is 3.14DCos(lat)/360. The number of tuples per degree rectangle can easily be arranged to reflect this falloff.

As a result of this arrangement, any given 1-degree latitude-longitude rectangle, that is, the curved rectangle with N-S sides equal to 1 degree of latitude and E-W sides equal to one degree of longitude, will be covered by a set of Terrain tuples, the number n in the set varying with latitude but equal to 4 X 3600Cos(lat). Thus for a 1-degree latitude-longitude rectangle, the number of 90-column-grid tuples required for Terrain is given by the following table:

Latitude	***Terrain tuples (max. colgrid value)***
0	*4 X 3600*
15	*4 X 3477*
30	*4 X 3118*
45	*4 X 2546*
60	*4 X 1800*
75	*4 X 932*

with the average number of tuples in a degree rectangle being close to 11,600. Since there are 360 x 360 degree rectangles on a sphere, the Terrain relation will have about 11,600 X 360 X 360, or 150 million tuples. Given a minimum of 3 bytes per grid element of a Terrain tuple, and 900 elements in a N-S column-grid and thus in a tuple, there are about 2700 bytes per tuple and thus 2700 X 150 million bytes, or about 400 gigabytes, in the relation Terrain.

Recalling that a 1 second grid element corresponds to about 16.5 meter square of surface, if we increase the resolution to 8.25 meters by using 0.5 second grid elements, the number of grid elements per tuple doubles to 1800, and the number of tuples for the globe also doubles to 300 million, with Terrain hence quadrupling to 1,200 Gigabytes.

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 the side of a 1-second grid element will vary, from place to place, by as much as 1%. Given a specific latitude, longitude and elevation with respect to the reference datum, a standard function [*trax (lat, long, elev)*] that returns the length of the grid sides can be stored with the Terrain relation as a virtual attribute (i.e. *trax()* takes up no storage space in a tuple).

Recall that for each degree of latitude, 4 distinct 900-grid-column tuples are used. Instead of 4 tuples, if a single 3600-grid-column tuple per degree of latitude were used, with 10,800 bytes per tuple, this would not do as well. The number of parallel N-S column-grid (and thus tuples) per degree of longitude falls from 3600 at the equator to 0 at the poles, as discussed already, that is, an average falloff of 4 tuples per degree of latitude. 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 1, for each quarter degree change in latitude on average. Were there only one tuple per degree of latitude, the falloff would be 4 tuples per N-S 1-degree latitude tuple on average, which would be too abrupt and would compromise the uniformity of the grid coverage of the planet. The 4 types of tuple per degree of latitude can be identified in Terrain by segment numbers 1 to 4, segment 1 being most southerly. A tuple of a Terrain relation of this design has the structure:

Terrain (lat long seg, colgrid, elev1 col1 comp1, elev2 col2 comp2, ...,
....., elev600 col900 comp900, trax ())

The terrain data in the array elements of a specific tuple, corresponding to Sound-to-North grid elements of a column-grid, are in left-to-right West-East order. For a specific latitude, longitude and segment number values, the colgrid value gives the number identifying of the N-S grid column. The maximum value for colgrid at any latitude is the number of 1-sec grid squares per degree of longitude. A colgrid value of 1 gives the most westerly column-grid, or corresponding most westerly tuple in the set of tuples for a specific latitude, longitude and segment value.

The lat-long values in any Terrain tuple are integer values for the N.W. 'corner' of a 1-degree curved rectangle, so that lat-long data of 43 215 means the curved rectangle between latitudes 43 and 42 North and between longitudes 215 and 214. 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. 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.E. quadrant of the 1-degree rectangle. The primary key is (lat long seg colgrid) .

An advantage of this structure is that any curved rectangle of data 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, this Terrain relation design solves four practical problems associated with a relation for Mars global data. The first problem solved concerns tuple size. Assuming 3 bytes of data per grid element, we need close to 2700 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 second problem solved is that the design accommodates the fact that as latitude increases, the surface length of a degree of longitude decreases, from about 59 km at latitude 0, to 0 km at latitude 90, and uses a fixed angular-coordinate grid element size (1 second of arc) that also stays constant in terms of surface length over the globe, to within a percent. The third problem solved is the geodetic problem of relating the size of a grid element to latitude, longitude and elevation for a triaxial ellipsoid. This is accomplished by placing a virtual attribute function *trax()* in Terrain. The forth problem solved by the design is that the way in which the data is gridded for storage in tuples is transparent to users constructing expressions in non procedural relational languages like SQL [8, 9], or expressions in more advanced languages [4, 10, 34], in order to extract data on the basis of features and feature properties, as will be apparent later; however, at the same time the design allows the user to extract data for any portion of any 1-degree rectangle.

Images and vistas

Users would will frequently want to have the grid of retrieved elevation and other data converted to an image of some kind, often a vista. Generation of images from the digital elevation data in Terrain will involve the use of *trax()*, cartographic projection techniques [22] that may involve a triaxial ellipsoid, and 3-D imaging methods. Fractal techniques can be used to enhance images where data is insufficient. The whole field is one of great research interest at the present time, but is largely outside the scope of this paper.

3. Relating Terrain data to terrain features.

Humans 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.

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 [2, 3].

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.

The database structure shown in Figure 1 includes relations to handle only crater, valley and mountain feature types, as this is a fair representation of the many feature-types and how they relate to Terrain.

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, ...)

where the primary key is fname, ftype gives the type of feature, e.g. crater, or valley, and clat and clong give latitude and longitude for the center point of the feature. FT is all key.

There will be in a 1:1 ISA relationship [3, 10, 34] 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 feature types to terrain, that is, relating the quite different relations, Crater, Valley, Mountain, and so on, to the relation Terrain. This problem is solved by enabling the n:m relationship that must exist between Feature and Terrain. 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 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).

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 (48.3,102.2). Suppose the crater lies inside the square with lat (48, 46) and long (102, 100) with center at (47.1, 101.2). The relevant tuples in the Feature, FT and Terrain relations would be:

fname	***ftype***	***clat***	***clong***
...	...	...	...
C3	crater	47.1	101.2
...	...	...	...
M6	mountain	48.3	102.3
...	...	...	...

Feature

fname	***lat***	***long***
...	...	...
C3	48	102
C3	48	101
C3	47	102
C3	47	101
M3	50	104
...	...	...
M6	49	101
M6	48	104
M6	48	103
M6	48	102
M6	48	101
M6	47	104
M6	47	103
M6	47	102
M6	47	101
...	...	...

FT

Lat	***long***	***seg***	***colgrid***	***elev1***	***colr1***	***text1***	...
...	...	...	...	...	...	...	
50	104						
...	...	...					
49	101						
...	...						
48	104						
...	...						
48	103						
...	...						
48	102						
...	...						
48	101						
...	...						
47	104						

Terrain

...	...
47	103
...	...
47	102
...	...
47	101

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 (47 103), 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 < 20 and clat > 0
            and clong > 0 and clong < 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 AFeature:

AFeature (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 - min long)/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 AFeature to Terrain, but only at the expense of either more complex or more tedious retrieval expressions. A rewrite of retrievals R1 and R2 below using Afeature instead of Feature and FT demonstrates this. The reader is invited to try the equivalent rewrite of R3 above.

R1: Get terrain data for the mountain Pavonis Mons.

```
Select * from Terrain
where lat <= (select maxlat from AFeature
          where fname = 'Pavonis Mons')
 and lat >= (select minlat from AFeature
          where fname = 'Pavonis Mons')
 and long <= (select maxlong from AFeature
          where fname = 'Pavonis Mons')
 and long >= (select minlong from AFeature
          where fname = 'Pavonis Mons')
```

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

```
Select AFeature.fname, Terrain.*  from AFeature, Terrain
where AFeature.maxlat >= Terrain.lat
  and AFeature.minlat <= Terrain.lat
  and AFeature.maxlong >= Terrain.long
  and AFeature.minlong <= Terrain.long
  and AFeature.ftype = 'mountain'
  and (AFeature.maxlat - AFeature.minlat)/2 <= 20
  and (AFeature.maxlat - AFeature.minlat)/2 >= 20
  and (AFeature.maxlong - AFeature.minlong)/2 <= 0
  and (AFeature.maxlong - AFeature.minlong)/2 <= 180
```

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 [3, 8, 9, 34] from AFeature, one equivalent to the relation Feature and the other to relation FT in Figure 1. This would enable expressions like the originals for R1, R2 and R3 to be used. However, FT could have as many as 50,000 tuples when a full roster of feature instances is stored, so that the necessary recomputing View-FT from AFeature each time a retrieval is executed seems like a waste; and View-FT will likely be needed in most retrievals given its strategic position linking Terrain with surface features.

An FT tuple is unlikely to exceed 20 bytes, so that the maximum size of FT is about 1 Mbyte, which is negligible compared with the size of Terrain (estimated earlier at 400 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 resulting reduction in complexity. Some readers may not agree, but in the rest of this paper we shall assume the Feature and FT relations as in Figure 1. From a wider perspective, this is just another example of the old trade-off of resources versus complexity.

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)

In the relation Crater, cname is the primary key, rimdiam1 and rimdiam2 are the outer and inner rim diameters; many other technical attributes beyond the scope of this paper can also be included.

In the relation Overlay, which is all key, 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 relation. 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)

In a Valley tuple vname is the primary key. 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.vname = 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).

Other 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. An example is the relation Mountain is included in Figure 1:

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

with 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 = 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 addtional tuples into Feature, one for every tuple in Plain, and to insert additional tuples into FT, one for every degree square 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 discussion of 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 for a regular angular-coordinate grid over the entire surface of Mars. The design is with respect to a triaxial ellipsoidal reference datum developed for Mars by USGS. At the core of the database is a 400-Gigabytes relation called Terrain that contains the terrain

data. It has a surface resolution of 16.5 meters, which can be increased to 8.25 meters at a cost of quadrupling the database size.

Data is recorded with respect to 1-second of arc grid elements. Each tuple of Terrain contains the latitude/longitude coordinates of a 1-degree curved rectangle, plus a column of 900 grid elements, called a column grid, for a N-S running column of grid elements within the 1-degree rectangle. The column-grid location within a 1-degree rectangle to which a tuple corresponds is determined by a segment and colgrid number in the tuple. The number of tuples in the E-W direction, and thus the number of N-S column-grids, varies from 900 per degree of longitude at the equator to 0 at the poles. This ensures a grid element is of constant size in meters, at least to within 1%, over the entire planet. Since a degree at the equator is close to 59 kilometers, a grid element is close to 16.5 meters square, giving a surface resolution of 16.5 meters

This Terrain relation design solves four practical problems associated with a relation for Mars global data. First it enable 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, by using a fixed angular-coordinate grid element size that also stays constant in terms of surface length over the globe. The third problem solved is the geodetic problem of how to relate the size of a 1-second grid element to latitude, longitude and elevation for a triaxial elippsoid in a convenient manner; it is solved by placing a virtual attribute function *trax()* in Terrain.

The remainder of the database has to do with relating Martian feature-type relations to Terrain. The fact that Terrain is structured so that there is a set of tuples per 1-degree square, the number in the set varying with latitude, is essentially transparent to users writing SQL expressions to retrieve Terrain data on the basis of specific features, even though are many different types of features, each requiring a distinct relation. This is the forth 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 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. FT is estimated to be only 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 [3, 31] in Terrain, so that it is possible to add additional feature-type relations in a modular manner. Since Feature-type relations are each in an 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 bus from Terrain, to which it is possible to attach any addition feature-type relation, regardless of any additional complexity due to recursive relationship participation. Three feature type relations were discussed, including the recursive Crater and Valley relations.

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.

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, about 5 X 5 degrees in size, in an area of a multiplicity of small features. To keep the database size down, yet enable the design to be tested, grid size could be increased to 6 seconds, for a resolution of about 100 meters. The size of the database prototype would then be a reasonable 30 Megabytes.

References

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

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

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

4. Bradley, J., 1996. Extended relational algebra for reduction of Natural quantifier COOL expressions, J. Systems Software, Vol 33, 87-100.

5. Browning B.A., Conway, B.J, Mueller, A.L, and D.J. Stanley, 1988. Exploiting Remote Sensed Images, Proceedings of a Royal Society Discussion Meeting, The Royal Society, London.

6. Carr M. H., et al, 1977. Martian impact craters and the emplacement of ejecta by surface flows, J. Geophys. Res. 82 (4), 35-64

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

8. Date. C. J. "Database Systems", 6th Edition, 1995.

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

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

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

12. Falcidieno, B., Pienovi, C., 1992. A feature-based approach to terrain surface approximation, Proc 4th Int. Symp. on Spatial Data Handling, Zurich, Switzerland, Vol 1, 190-199.

13. Floriani L., Gattorna, P., 1994. Spatial queries on a hierarchical terrain model, Proc 6th Int. Symp. on Spatial Data Handling, Edinburgh, Scotland, Vol 2, 819-834.

14. Fournier A, Fussel, D, and Carpenter, L., 1982. Computer rendering of stochastic models, Comm. of the ACM, 25, 371-384.

15. Fishman, B., and Schachter, B., 1980. Computer display of height fields, Computers and Graphics, Vol 5, 1980, 53-60.

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

17. Greeley, R., Bateson, R., 1997. "NASA Atlas of the Solar System", Cambridge University Press.

18. Gulick, V.C, Baker, V.R., 1990. Origin and evolution of valleys on Martian volcanoes, J. Geophys Res. 95(14), 235-344.

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

20. Kufoniyi, O, Pilonk, M, 1994. A vector data model integrating multitheme and relief geoinformation, Proc 6th Int. Symp. on Spatial Data Handling, Edinburgh, Scotland, Vol 2, 1061-1071 .

21. Lathrup, R. G., 1992. Identifying structural self-similarity in mountainous landscapes, J. of Landscape Ecology, Vol 6., 233-238.

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

23. Moore, P., 1977. "Guide to Mars", Lutterworth Press, London.

24. Mueller, I. I., Rapp, R. H., 1989. Horizontal and vertical geodetic datums, in "Reference Frames in Astronomy and Geophysics", Kovalevsky, J., Mueller, I. I., Kolaczek, B. (Editors), Kluwer Academic Publishers, London.

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

25. NOAA and NASA, 1987. Space-based Sensing of the Earth: A Report to Congress, Washington D.C., U.S. Government Printing Office.

27. NASA, 1987. Altimetric systems, NASA Earth Observing System Reports, Vol 2h.

28. Ramesh, R., Babu, A. J. G., Kincaid, J. P., 1989. Index optimization: Theory and experimental results, ACM Transactions on Database Systems, 14(1), 41-74.

29. Samet, H., Aref, W., 1995. Spatial data models and query processing, in "Modern Database Systems", Kim, W. (Editor), ACM Press/Addison Wesley, 338-360.

30. Saupe, D., 1988. Algorithms for random fractals, in "The Science of Fractal Images", H. Peitgen (Editor), Springer-Verlag, New York.

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

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

33. Sperry, S. L. Novak, K., 1992 Integrating digital photgrammetry and GIS. URISA Annual Conf. Proc., Vol 2, 36-44.

34. Stonebraker, M., Anton., J, and Hanson, E., 1987. Extending a database systems with procedures, ACM Trans. on Database Systems 12(3), 350-376.

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

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

37. Voss, R. M., 1988. Fractals in nature, from characterization to simulation, in "The Science of Fractal Images", H. Peitgen (Editor), Springer-Verlag, New York.

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

□