Large Haptic Topographic Maps: MarsView and the Proxy Graph Algorithm

Sean P. Walker[*]
Computer Science Department
Stanford University
Stanford, CA

J. Kenneth Salisbury[†]
Depts of Computer Science and Surgery
Stanford University
Stanford, CA

Abstract

In this paper we develop an interactive 3D browser for large topographic maps using a visual display augmented by a haptic, or force feedback, display. The extreme size of our data files (over 100 million triangles) requires us to develop the "proxy graph algorithm", a new haptic contact model. The proxy graph algorithm approximates proven virtual proxy methods but enhances the performance significantly by restricting the proxy location to the edges and vertices of the object. The resulting algorithm requires less computation and reduces the average number of collision detection operations per triangle that the proxy crosses during each haptic update cycle. We also develop a collision detection algorithm optimized for our heightfield dataset.

Our "MarsView" software enables hands-on interactive display of visual and geologic data with polygon counts in excess of 100 million triangles using a standard PC computer and a commercial haptic interface. MarsView's haptic user interface allows the user to physically interact with the surface as they pan it around and zoom in on details. The hybrid system renders complex scenes at full visual and haptic rates resulting in a more immersive user experience than a visual display alone.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality H.5.2 [Information Interfaces and Presentation]: User Interfaces—Haptic I/O

Keywords: haptics, interface, Mars, topological map, large datasets, texture, virtual proxy

1 Introduction

Within the past few decades the technology has become available to explore and accurately map most of our world and Mars. A vast quantity of topographic data is now available from satellite and surface rover surveys. Mars, especially, has been a focus of recent study and detailed maps are now available for the first time which include close to a billion data samples [NASA n. d.].

With the new availability of geographic data comes a need to efficiently browse through it and view it. While some work has been done in this area [Stoker et al. 1999; Nesbitt et al. 1997], there is still a need for highly interactive means of organizing and interacting with an ever-increasing body of topographic information streaming in from our planetary science probes. Such a tool would allow the scientific community to better understand the geography of remote planets and allow the general public to better explore our world and others.

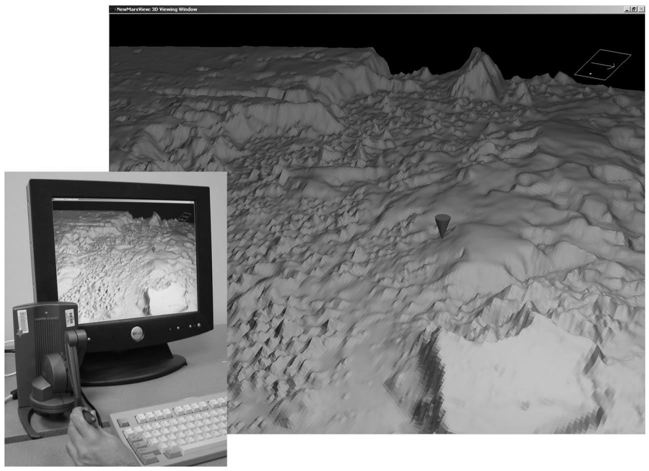


Figure 1: MarsView in action.

This paper presents a first attempt at augmenting a visual topographic map display system with force display, or haptics. Haptic interface devices use force feedback techniques to allow a human user to physically interact with virtual objects – effectively allowing physical manifestations of objects modeled in the computer to be touched [Salisbury and Srinivasan 1997]. Adding force feedback to a topographic map viewer allows the user to actually "touch" the surface and feel the high resolution data as texture. In addition, a haptic user interface allows more intuitive three-dimensional physical interaction providing a more immersive user experience.

Haptic rendering, much like visual or graphic rendering, requires a physical model that can be queried to produce the appropriate stimulation. Haptic rendering, however, requires update rates (on the order of 1 kHz) significantly higher than required in graphics; it also focuses on the real-time physical modeling of forces and motion rather than light. One of the significant challenges we addressed was how to maintain sufficiently high update rates when rendering datasets with a very high density of geometric information–such as that encountered in planetary surface maps. A significant accomplishment of our work has been the creation of a new class of algorithms which rely on the *proxy graph contact model* reported herein.

The proxy graph contact model approximates the proven virtual "proxy" methods such as the "god-object" method devised by Zilles [1995]. By restricting the proxy location to the vertices and edges of a single seamless non-intersecting triangular mesh a speedup of several orders of magnitude can been accomplished

[*]e-mail:spw@cs.stanford.edu
[†]e-mail:jks@robotics.stanford.edu

over previous methods. This allows extremely large topographic maps of over 100 million triangles to be haptically rendered on standard IBM PC compatible computers using commercial haptic devices such as SensAble Technology's PHANTOM line [Massie and Salisbury 1994] or the DELTA line of devices from Force Dimension [Grange et al. 2001].

Using the proxy graph algorithm we created MarsView (depicted in Figure 1), an application that allows the user to haptically and graphically view large topographic maps of Mars or Earth. Within MarsView, a map may be touched and manipulated using a novel haptic user interface. MarsView allows the graphic display to be run at a lower resolution than the haptic display to keep frame rates responsive yet allow the hidden data to be felt as haptic texture. In addition, a haptic user interface gives the surface convincing physical properties while it is examined at different levels of detail with zooming and panning operations.

The remainder of this paper is organized as follows: Section 2 discusses the advantages and limitations of using existing proxy haptic contact models with large datasets. Section 3 describes the general proxy graph algorithm in detail along with some optimizations we were able to make when using a heightfield data representation. Section 4 describes the MarsView application including its haptic user interface. Sections 5 and 6 discuss future work and conclusions respectively.

2 Previous Work

This section begins by describing haptic contact models and then focuses on proxy based contact models. Then, it combines common themes from the two most prevalent proxy algorithms into a general proxy contact model algorithm and discusses the merits and limitations of that algorithm.

2.1 Haptic Contact Models

A haptic simulation requires three basic components: the haptic device (which exerts forces on the user and senses position in 3D), the software representation of the virtual objects, and the contact model. Most haptic research has focused on the contact model because it provides the connection between the haptic device and the virtual scene, generating forces based on the current position of the haptic device at least a thousand times a second to give the user the impression of touching a virtual object.

Although many contact models exist, they can be classified into one of two types: those that use the previous position along with the current position of the haptic device (which we call "historical methods" because they depend on prior state information), and those which only use the current position. While non-historical contact models are generally easier to implement, the lack of knowledge about the direction of penetration of the object often causes artifacts or unstable behaviour (see [Zilles and Salisbury 1995] for a complete discussion).

Our work focuses on historical contact models, specifically the "god-object" method devised by Zilles [1995]. Zilles' method belongs to a class of methods which use a "proxy", or a virtual object that follows the haptic endpoint but cannot penetrate the objects in the scene. Proxy based methods are the most popular haptic contact models because they are relatively easy to implement, are extremely stable, and they produce reasonably realistic haptic forces.

2.2 Proxy Contact Models

The key element of the proxy contact model family is the virtual object, or proxy, that represents the haptic interface in the virtual scene. The proxy, both frictionless and massless, is connected to the haptic endpoint through a simple linear spring model as shown in Figure 2. Since the proxy cannot penetrate virtual objects, touching an object will cause the spring to stretch and apply a corresponding force on the user through force feedback. At the same time, the proxy will slip along the surface until it either falls off or reaches the point that is closest to the haptic endpoint (effectively moving to a constrained local minimum on the surface through gradient descent). This minimization operation, which is similar to the way that water will run downhill and collect in low areas, is at the heart of all proxy contact model algorithms.

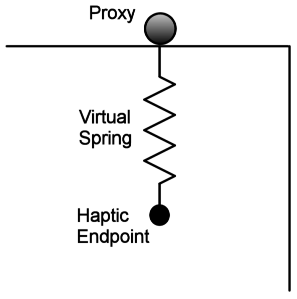


Figure 2: The virtual spring connecting the proxy and the haptic endpoint.

Existing proxy contact models generally distinguish themselves in one of three areas: object representation, proxy shape, or minimization technique. Often, the first two areas are selected to make distance minimization as fast and easy as possible. The majority of proxy models use a polygonal object representation although some use other representations such as implicit functions [Salisbury and Tarr 1997] or NURBs [Thompson II et al. 1997]. To simplify minimization, proxy shape is generally restricted to a point or, in some cases, a sphere [Ruspini et al. 1997]. Although it is beyond the scope of this paper, many contact models also allow for additional features such as friction and force shading (polygon normal interpolation to make the surface feel smoother) [Morgenbesser and Srinivasan 1996].

Both of the most popular proxy contact models, Zilles' god-object algorithm [Zilles 1995; Zilles and Salisbury 1995] and Ruspini's virtual proxy algorithm [Ruspini et al. 1997], use a polygonal object representation. The main difference between the two is that the god-object algorithm implements a point proxy while the virtual proxy algorithm implements a sphere proxy. Minimization in both algorithms is actually quite similar since Ruspini minimizes in configuration space (where each object is grown by the radius of the sphere) using a point proxy. For both algorithms each polygon contributes a constraint plane that limits the motion of the proxy – while Zilles uses the actual polygon planes, Ruspini uses the tangent planes of the configuration space objects.

2.3 General Proxy Algorithm

A generalized proxy algorithm can be used to describe the behavior of both the god-object algorithm and the virtual proxy algorithm. The basis of the general algorithm is a list of constraints, initially empty, that restricts the motion of the proxy as it moves towards the haptic endpoint location. During each haptic update, the proxy attempts to move towards a goal location (initially the haptic endpoint) until it either reaches the goal or collides with another polygon that adds another constraint plane, resulting in a new goal location. The goal location, or constrained local distance minimum, may be calculated from the set of active constraints using Lagrange

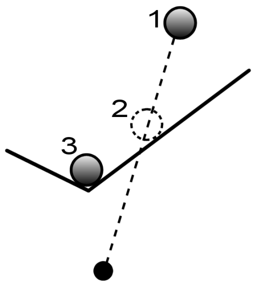


Figure 3: General proxy minimization: proxy starts at 1 and moves towards haptic endpoint, colliding with surface at 2, and slides to the local distance minimum at 3.

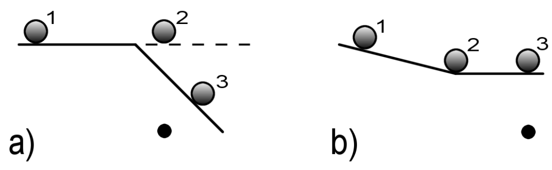


Figure 4: General proxy algorithm constraint release limitations: proxy starts at 1 and terminates at 2 instead of 3, causing either a) floating proxy or b) stuck proxy.

multipliers as described in [Zilles and Salisbury 1995]. The whole minimization process will terminate within three iterations (with a point proxy) because three constraints will restrict the proxy motion in three dimensions to a single point. Figure 3 illustrates the process in two dimensions for clarity.

On small surfaces, since the haptic update rate is typically 1 kHz, the proxy will rarely cross over more than one triangle during each haptic update. On large surfaces the proxy may need to cross over 10 triangles or more, yet the general proxy algorithm cannot actually cross over more than one triangle because it does not release constraints once they are added to the active constraint list. There are two problems that may result from this: Figure 4a demonstrates a 'floating' proxy that results if the surface slopes away from the current constraint and Figure 4b demonstrates a 'stuck' proxy that can occur if the surface slopes toward the current constraint. These small artifacts will be unnoticable on small surfaces because of the fast update rate but will cause an apparent viscosity effect on larger surfaces due to limited proxy speed. The constraint release problem can be remedied relatively easily by modifying the general proxy algorithm to release constraints once the proxy is no longer in contact with the corresponding triangle.

After modifying the general proxy algorithm to include constraint release we found that our implementation was at least an order of magnitude too slow. Since the Martian model contains over 100 million triangles, a fast swipe across the surface may pass over approximately 10,000 triangles in about half a second. With a haptic update loop of 1 kHz this required 20 triangles to be processed within a millisecond. The modified algorithm has a running time that is linear in the number of constraints that are processed (and likewise linear in the number of triangles that the proxy crosses because each triangle adds a constraint). Since each iteration requires at least one collision detection operation and a Lagrangian computation, we found our computers simply did not have the computational power to do tens of iterations within one millisecond.

We also found that the majority of general proxy algorithm computation time was spent in collision detection. Collision detection algorithms have significant overhead even if optimized for haptics (such as H-COLLIDE [Gregory et al. 2000]) or for the specific problem (such as the heightfield optimizations proposed in Section 3.5). Collision detection on extremely large models is especially difficult and slow because computation increases superlinearly as the number of polygons increases linearly.

We developed the proxy graph contact model to reduce the average amount of computation required to process each triangle by primarily reducing the number of collision detection operations. The proxy graph algorithm restricts the proxy location to be on the vertices of the object during intermediate minimization steps within a haptic update cycle to avoid collision detection as long as the minimization path is in contact with the surface. The resulting algorithm is stable and will produce almost the same results as the general proxy contact model.

3 Proxy Graph Approach

In this section we will describe the proxy graph contact model. First, we give an overview of the proxy graph algorithm by intuitively describing its operation and briefly highlighting its advantages and disadvantages over previous proxy algorithms. Then we describe the algorithm in detail and discuss its stability and termination properties. Finally, we highlight the specific optimizations we applied when we used the proxy graph algorithm with heightfield topographic map datasets.

3.1 Overview of Proxy Graph Contact Model

The proxy graph contact model requires a seamless, non-intersecting triangular mesh consisting of a connected graph of faces, edges, and vertices. More specifically, no two triangles in the mesh may intersect or be positioned next to each other without sharing an edge in the mesh. While the topological map application outlined in this paper uses a heightfield object, the proxy graph algorithm is applicable to any seamless non-intersecting triangular mesh including closed objects or multiple independent objects. Generally the triangular mesh will be one sided – meaning the proxy can pass through the back of triangles – but it is not necessary for correct algorithm operation.

The proxy graph contact model achieves faster performance than the general proxy model by alternating between two modes of operation as shown in Figure 5a: free space minimization, or minimization out of contact with the surface, and minimization constrained by the surface. The algorithm begins in free space minimization, moving the proxy along the direct path to the goal, or haptic endpoint location, until it reaches the goal or collides with an object. If the proxy collides with an object, the algorithm switches to constrained minimization, an optimized gradient descent along the edges of the triangular mesh as depicted in Figure 5b. Essentially, the proxy moves from vertex to vertex by following the edges that decrease the distance between the proxy and the goal the fastest. As the proxy moves along each edge, each of the two adjacent triangles are checked for local distance minima and the direct path to the goal is checked to see if free space minimization can be restarted.

The main advantage of the proxy graph algorithm over the general proxy algorithm is a *significant* decrease in the number of collision detection operations. This is achieved by minimizing along the triangular mesh of the surface using simple gradient calculations whenever the proxy is in contact with the surface. The result is an algorithm that has a very low time constant even though it still has a running time linear in the number of triangles crossed during each haptic update.

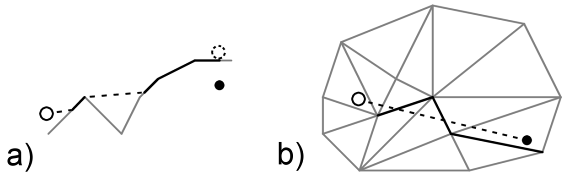


Figure 5: Proxy graph behavior: a) side view showing alternation between free space minimization (dashed lines) and constrained minimization (solid lines) b) overhead view of a flat triangulation to demonstrate the path the general proxy takes (dotted line) and the path the proxy graph algorithm takes (solid line).

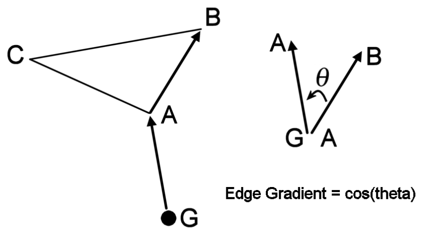


Figure 8: Illustration of the edge gradient computation for edge AB and goal location G: Edge gradient = $cos(\theta) = \frac{B-A}{||B-A||} \cdot \frac{A-G}{||A-G||}$

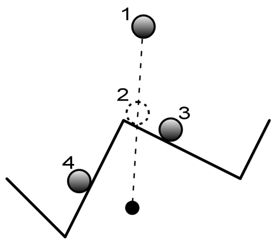


Figure 6: An illustration of how a small error from vertex tracing causes an alternative local minimum to be found. Proxy at 1 moves towards endpoint, colliding with surface at 2, and instead of moving to local minimum at 3 it starts at the vertex and moves to 4 because that edge initially decreases distance faster.

Although the proxy graph is effectively equivalent to the general proxy algorithm there are some drawbacks. First, because the algorithm alternates between free space minimization and constrained minimization (without collision detection) the algorithm will only work with non-intersecting triangular meshes, not a serious drawback with physically consistent data. Second, because the proxy location is restricted to vertices during constrained minimization, there will be small errors between the ideal minimization path and the actual path. These small errors can cause the proxy to slip into the area of a different local minima (Figure 6) or collide with different portions of the surface when it takes a slightly different path through free space. The user will generally not notice the difference though because the errors will be small and it is difficult to distinguish between the proper behavior and the actual behavior.

3.2 General Proxy Graph Algorithm

The proxy graph algorithm runs during each haptic update cycle to compute a new proxy location from the current haptic endpoint position and the previous proxy position – this new location is then used to compute the virtual spring force applied to the user. Figure 7 depicts a complete flow chart outlining the algorithm. As with all proxy algorithms, the purpose of the proxy graph contact model is to minimize the distance between the new proxy location and the haptic endpoint location, or goal, while being constrained by the surface. Two different types of minimization occur: free space minimization and constrained minimization.

Free space minimization occurs whenever the path from proxy to goal is unobstructed by the surface. A single collision test is used to advance the proxy until it either reaches the goal or contacts a triangle in the scene. A collision with a triangle begins constrained minimization, or gradient descent along the edges of the triangular mesh, beginning with the collision triangle vertex that is closest to the goal. Constrained minimization will continue until a local distance minimum is found at a vertex, edge, or adjacent triangle. Free space minimization may also resume if the path from the current vertex to the goal is unobstructed by the adjacent faces of the vertex.

The algorithm spends in the majority of its time in constrained minimization so it needs to be made as fast as possible. This is accomplished by restricting the proxy location to vertices during gradient descent and greedily choosing to trace along the edges that decrease the distance to the goal the fastest. At each vertex, we choose the edge with the smallest edge gradient, or dot product between the normalized vector along the edge and the normalized vector from the goal to the vertex (this is effectively the cosine of the angle between the two vectors, as depicted in Figure 8). An edge gradient is negative if a small step along the edge will decrease the distance to the goal (the more negative the edge gradient, the faster distance will decrease). If the gradients of all edges are positive then no step along any edge or face will decrease the distance to goal; the current vertex is the local minimum and final proxy location. Otherwise, the edge with the most negative edge gradient will decrease the distance to the goal fastest – the proxy should move to the vertex at the end of this edge as long as it is closer to the goal than the current vertex.

If the end vertex of an edge with a negative edge gradient happens to be further from the goal than the current vertex we know that the distance from proxy to goal as the proxy moves along the edge must have initially decreased and then increased. This means that there is a local minimum on the edge and possibly other local minima on adjacent faces. The final proxy location will be the edge or face local minimum that is closest to the goal.

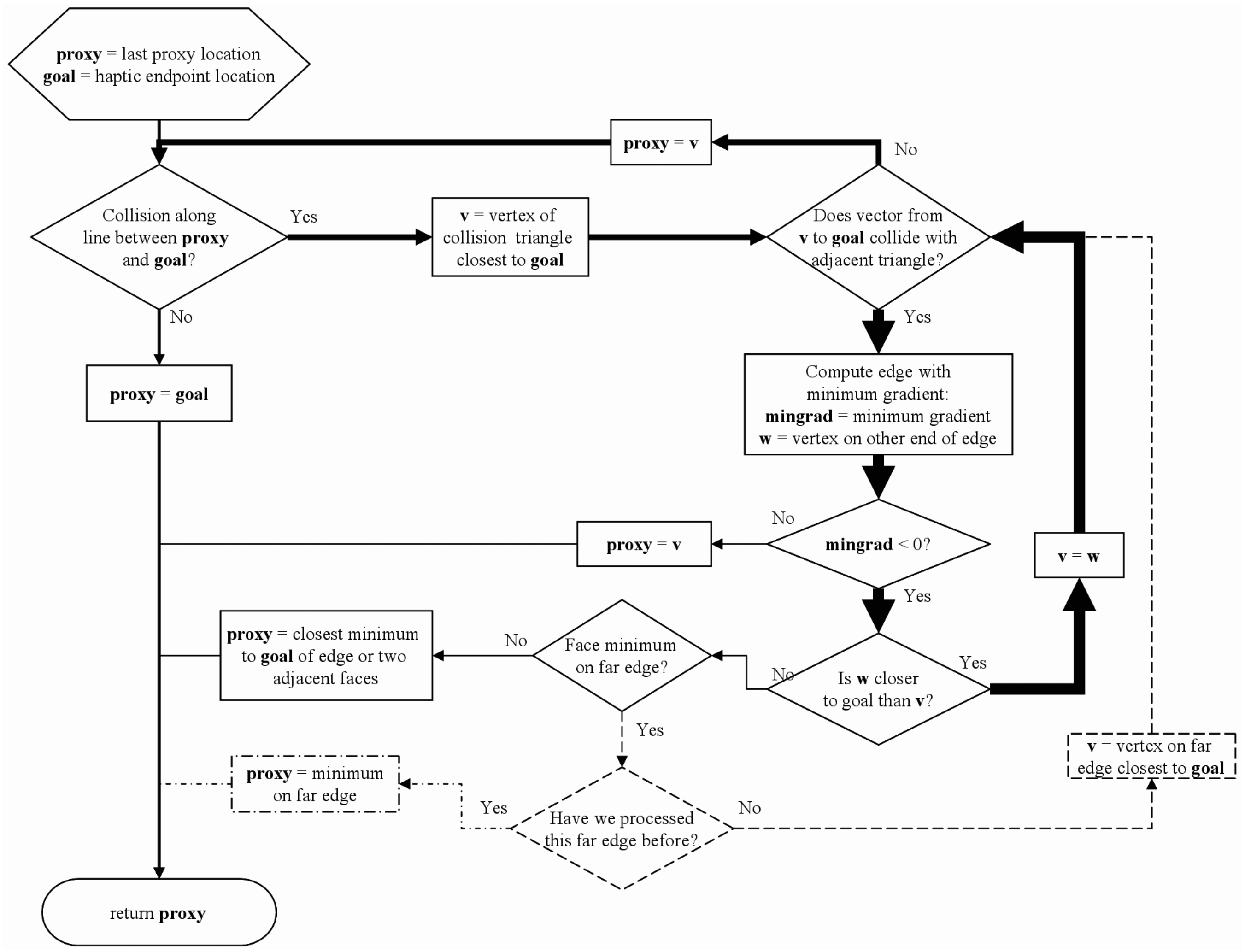
We can find the local minimum on a face or edge by computing the orthogonal projection of the goal location onto that face plane or edge line and verifying the projected point is within the boundaries of the face or edge. The orthogonal projection is the unique point on the face or edge where the vector from that point to the goal is perpendicular to the face plane or edge line. The orthogonal projection of a point G onto an edge starting at vertex A with a unit length direction vector D is:

$$P_{edge} = A + [D \cdot (G - A)] * D \tag{1}$$

And the orthogonal projection of G onto a face plane going through vertex A with unit length normal N is:

$$P_{face} = G - [N \cdot (G - A)] * N; \tag{2}$$

The algorithm we have outlined will generally find the local distance minimum the majority of the time. Occasionally though, constrained minimization using gradient descent will fail due to a far edge crossing, described in the next section.



proxy = last proxy location
goal = haptic endpoint location
proxy = v
No
Collision along line between proxy and goal?
Yes
v = vertex of collision triangle closest to goal
Does vector from v to goal collide with adjacent triangle?
No
Yes
proxy = goal
Compute edge with minimum gradient:
mingrad = minimum gradient
w = vertex on other end of edge
No
proxy = v
mingrad < 0?
v = w
Yes
proxy = closest minimum to goal of edge or two adjacent faces
No
Face minimum on far edge?
No
Is w closer to goal than v?
Yes
Yes
v = vertex on far edge closest to goal
proxy = minimum on far edge
Yes
Have we processed this far edge before?
No
return proxy

Figure 7: A flowchart depicting the proxy graph algorithm. The solid lines show the common paths of execution (larger lines show which paths are taken most often) and the dotted lines show paths that are very rarely taken.

3.3 Far Edge Crossing Problem

During constrained minimization there will sometimes be a path across a face that decreases distance to goal faster than either of the adjacent edges but the orthogonal projection of the goal, or projected minimum, will not be within the edges of the triangle. As shown in Figure 9, a local minimum will not be found on the face and the algorithm outlined so far would terminate with a local minimum on the adjacent edge. This would create a noticable artifact though, since the desired path of the proxy would be to reach some other local minimum by moving across the face to the far edge of the triangle.

The far edge crossing problem may be better understood by analyzing distance minimization on the plane of each face adjacent to the current vertex. Since the projected minimum is the point on the plane that is closest to the goal any path along the face that decreases distance to the goal must also decrease distance to the projected minimum. Or, alternatively, the path on the face that decreases distance to the goal the fastest is the direct path from the current vertex to the projected minimum. This fact allows us to explore minimization paths along a single face in two dimensions instead of three dimensions. It also means that, in the case of the far edge crossing, the point on the face that is closest to the goal is actually a point on the far edge since that point is also closest to the projected minimum.

Far edge crossings may be easily detected by testing if the projected minimum is within the lines of the two edges adjacent to the current vertex but outside of the far edge. Since the triangle minimum is on the far edge, the proxy graph algorithm can avoid the far edge crossing problem by continuing constrained minimization with the vertex on the far edge that is closest to the goal. There will be at least one path from this new vertex that will continue to decrease distance to the goal by moving towards the far edge minimum, even though the new vertex is further from the goal than the current vertex.

It is possible that there is an edge leaving the new vertex that has a smaller edge gradient than the far edge. In this case, while the ideal minimization would have crossed the far edge, the proxy graph minimization will search in a different area and reach some other local minimum. The user will generally not notice this because it is difficult to predict proxy behavior in these situations. But, the fact that the new vertex is further from the goal than the current vertex makes it possible for the new path to actually lead to the original far edge crossing vertex, causing an infinite loop. The

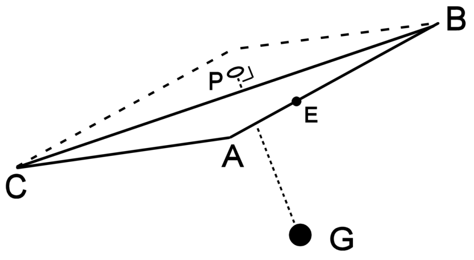


Figure 9: Illustration of the far edge crossing problem. Triangle ABC is shown with its corresponding face plane where A is the current vertex and AB is the edge with minimal gradient. P is the projected minimum resulting from orthogonally projecting goal position G onto the face plane. Since P is not within the boundary of triangle ABC a local minimum would not be found on the face and the edge minimum E would be the result if far edge crossings were not detected (even though the path across the triangle would result in a smaller distance).

proxy graph algorithm avoids these extremely rare loops by maintaining a small list of vertices where far edge steps have occurred – if the same vertex is encountered twice it terminates with the proxy at the face minimum found on the far edge.

We can demonstrate correct algorithm operation in spite of far edges by enumerating the possible projected minimum locations in the plane of a triangle and show that each will be handled correctly. Figure 10 depicts the face plane of a triangle ABC where vertex A is the current location of the proxy and edge AB has a negative edge gradient that is smaller than all others leaving A. Three guidelines are drawn in the diagram to split the face plane: 1 is the angular bisector of angle CAB, 2 is the perpendicular bisector of edge AB, and 3 is a line normal to AB denoting projected minimum locations where the edge gradient for AB cease to be negative. Using these guidelines, the projected minimum may exist in one of four areas where either a local minimum is found correctly or the algorithm will continue searching from another vertex. If the projected minimum is in area (a) vertex B will be closer to the goal then vertex A and the algorithm will continue searching at vertex B. If the projected minimum is in area (b) then the projected minimum is not actually on triangle ABC – there will either be a local minimum on edge AB or a local minimum on the adjacent face (which can be treated with the same method as this face). A projected minimum in area (c) will produce a true local minimum on face ABC because it is within its boundaries. A projected minimum in area (d) causes a far edge crossing, but continuing searching from B or C will allow the minimum to be found.

It is important to realize that far edge crossings, although a relatively uncommon occurrence, become more common as the triangular mesh becomes more uneven. Since area (d) in Figure 10 will grow if angle CAB increases and/or edge AB becomes longer in relation to edge AC, it is best to keep all edge lengths similar sizes and all angles less than 90 degrees. This can be accomplished by retesselating the triangular mesh as needed if it is uneven. In the case of this paper, the topographic data did not require any retesselation.

3.4 Stability and Termination

As with any haptic algorithm it is important to demonstrate that the proxy graph contact model is stable. The best way to ensure stability is to show that small changes in endpoint position will not be amplified into disproportionately large forces which can cause

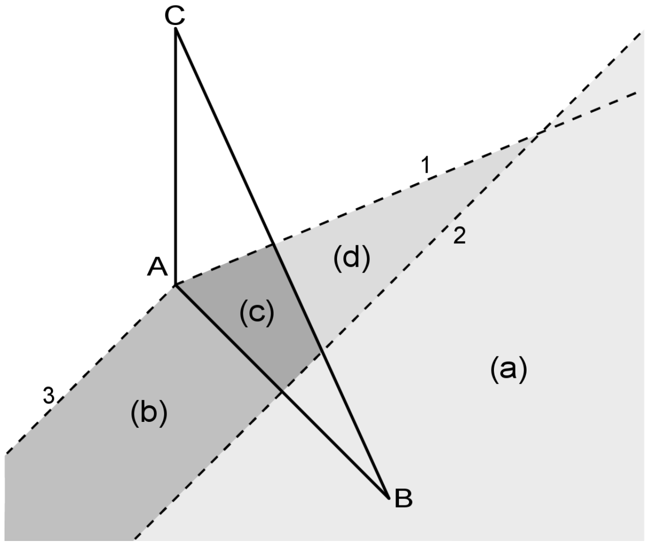


Figure 10: The subdivision of the triangle plane showing areas where the projected local minimum may lie for use in analyzing the behavior of the proxy graph algorithm.

vibration and/or unpredictable behavior. For the proxy graph algorithm, this is simply a case of showing that the final proxy location will be at a local distance minimum at the end of each haptic update. If the proxy always terminates at a local minimum then the distance between the proxy and the goal will never increase faster than the haptic endpoint moves. Since the virtual spring constant determines how much the force increases for a corresponding increase in distance, the resulting simulation will be stable as long as the spring constant is within the stable range for the haptic device.

The proxy graph algorithm always terminates at a local minimum on a vertex, edge, or face as demonstrated in Figure 10. In the rare case of a far edge crossing, the proxy graph algorithm may not stop at the same minimum as the general proxy algorithm, but it will stop at a local minimum. Likewise, in the extremely rare case of a far edge loop the algorithm will terminate at the minimum on the far edge, but our testing has shown that this does not happen enough to influence stability.

Termination of the proxy graph algorithm is also relatively easy to show because every step the algorithm takes will decrease the distance to the goal – except for far edge steps. The algorithm avoids infinite loops due to far edge steps by maintaining a list of far edge steps that occurred during that haptic update and terminating if a loop is detected. As a result, the proxy graph algorithm will always terminate.

3.5 Heightfield Dataset Optimization

Although the proxy graph algorithm will work with any non-intersecting triangular mesh, our implementation uses a 2 1/2 dimension data set, or height field. A height field, commonly used for topographic maps, consists of evenly spaced elevation values in a two-dimensional grid. The surface is triangulated by adding a diagonal edge which splits each grid cell into two triangles. As a result, a 2D array of floating point numbers implicitly encodes a triangular mesh – efficiently storing very large datasets (exceeding one hundred million triangles on our test machines). Also, additional collision detection optimizations are possible because heightfields are vertically monotone (each horizontal point has only one unique surface point directly above or below it).

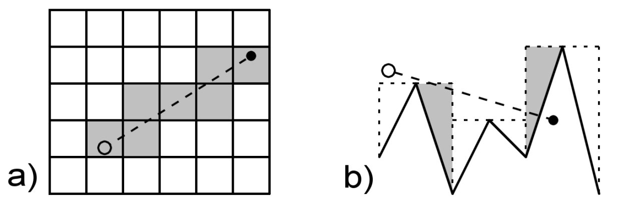


Figure 11: Heightfield collision detection optimizations: a) overhead view: only cells along the path from proxy to goal are searched for collisions b) side view: cell culling – triangle tests are only needed in those cells (shaded) where the collision path is lower than the highest vertex

Even though the proxy graph algorithm reduces the overall number of collision detection operations, it is still important to make collision detection as fast as possible by capitalizing on the special structure of the data. Our main optimization is to project the path that is being tested for collisions onto the horizontal plane as shown in Figure 11a. Since each point on the horizontal plane corresponds to only one point on the surface we can search the height field grid cells in order using a very fast digital differential analyzer technique. As the algorithm traverses the grid it tracks the current height of the path where it enters and leaves each cell and whether that path is above or below the surface. This allows a quick rejection of collisions in most cells because the entire path will be entirely above the surface as shown in Figure 11b. The triangles in the unculled cells can then be individually tested for collisions – this can be accomplished efficiently by testing whether the path enters the triangle cell above the triangle plane and leaves the cell below the triangle plane.

The special properties of the height field data structure also allow optimization of the transition from constrained minimization to free space minimization during the proxy graph algorithm. The free space test determines if the path from the current vertex to the goal is obstructed by an adjacent triangle. Due to the structure of the height field only one adjacent triangle may obstruct the path to the goal – in fact, we can easily determine the triangle to test by projecting the path to the horizontal plane and using the fixed structure of the triangulation to quickly select the correct triangle. Since the proxy is assumed to be slightly above the surface during constrained minimization, it is a simple test to see if the goal is above the plane of the triangle, and hence the path is unobstructed.

4 Application Example

In this section we describe the MarsView software that we developed in order to haptically view large topographic maps. In addition to haptically rendering the surface with the proxy graph algorithm, MarsView implements a haptic user interface to aid the user in exploration.

4.1 Hardware

MarsView was implemented to run on an IBM-compatible PC using the PHANTOM series of haptic devices made by SensAble Technologies (we used both the PHANTOM Desktop and the PHANTOM Premium 1.5/6 DOF). PHANTOM haptic interfaces provide high-performance 3D positioning and force feedback plus a 3 degree-of-freedom orientation sensing gimbal. PHANTOM support is provided by SensAble Technologies' PHANTOM Device Drivers Version 3.1 and Ghost SDK Version 3.1 at a near real-time

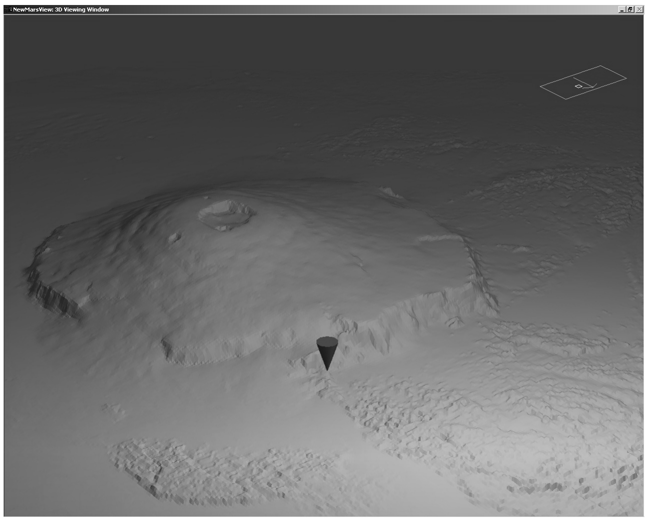


Figure 12: A typical MarsView window depicting Olympus Mons. The user interacts with the surface using the red cursor in the foreground. A small overview display appears in the upper right hand corner.

1 KHz servo rate under Microsoft Windows 2000. Our implementation uses the gstForceField class to directly display the forces from our contact model code on the PHANTOM. Table 1 describes the two systems used for testing: a laptop and a high-performance dual processor desktop PC.

4.2 User Interface

MarsView was designed to combine graphic and haptic display of topographic maps in an intuitive and easy to use interface. We augmented a standard OpenGL graphic display system with haptic force feedback to allow the user to both feel the topography of the surface and navigate within the virtual Martian world. MarsView maintains a high framerate by dynamically dropping samples as needed, although the speed of the proxy graph algorithm allows the full dataset to be displayed haptically (unseen data is felt as texture). A typical MarsView window is shown in Figure 12.

The user's main element of interaction with MarsView is a cursor representing a point in 3D space. The cursor can be used to touch the surface or, in conjunction with the PHANTOM stylus button (or space bar), to change the current mode of interaction and manipulate the surface in relation to a fixed camera. Since each mode – normal, pan, or zoom – results in radically different force interaction, the cursor shape changes to give a visible indication of the current mode.

Normal mode, the most common mode of operation, allows the user to directly touch and explore the visible portion of the surface using the proxy graph contact model. In normal mode, the cursor changes from a sphere when not touching the surface to a downward facing cone when in contact with the surface. Additionally, MarsView renders four haptic constraint walls to keep the user within the PHANTOM workspace. MarsView will remain in normal mode until the stylus button is pressed: if the button is pressed while in contact with the surface it will enter pan mode and if the button is pressed while not in contact with the surface it will enter zoom mode.

Pan mode simulates grabbing the surface with the cursor (which changes to a vertically flattened sphere) and pushing or pulling the surface horizontally while the vertical position is held constant. The top sequence of frames in Figure 13 shows surface panning. The

Type	Brand/Model	Processor	RAM
Laptop	Toshiba 3000-s353	900 MHz Pentium III	384 MB SDRAM
Desktop	Dell PWS530	Dual 1.7 Ghz Pentium III	1 GB RDRAM

Table 1: Test System Specifications

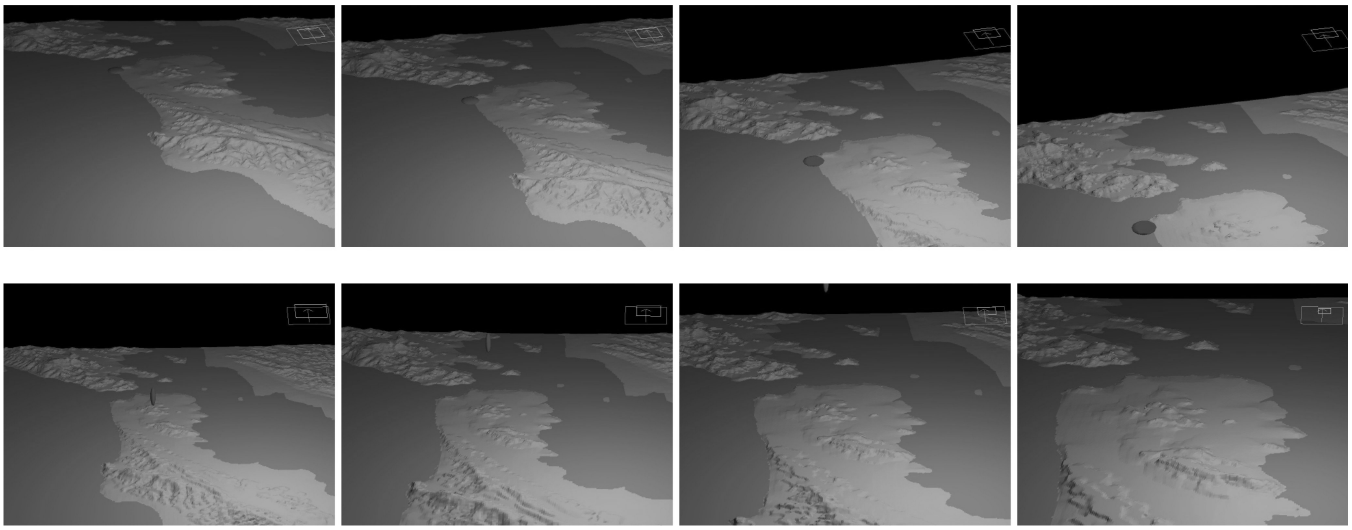


Figure 13: Sequences of images of the San Francisco area showing panning (top) and zooming (bottom).

panning simulation includes inertia (any change in velocity is resisted by an inertial force) and momentum (the surface will continue moving if the user releases their hold while the surface is in motion). Momentum portrays a strong feeling of virtual mass and allows the user to explore the surface as it moves. Walls are also implemented in panning mode to keep haptic interactions within the stable area of the PHANTOM workspace.

Zoom mode simulates grabbing empty space and ratcheting up or down, while the horizontal position is held fixed, to move the surface towards or away from the camera. The cursor changes to a vertically elongated sphere while in zoom mode. To guide the user to discrete zoom values, small force feedback "detents" are created using a weak linear force. Additionally, the surface can be rotated around the vertical axis through the cursor by rotating the PHANTOM stylus. The bottom sequence of Figure 13 depicts a surface being zoomed.

Since the MarsView user interface combines a number of haptic forces, it is important to make sure that they are mixed in a stable way. For instance, if two stable forces (such a wall constraint force and a surface contact force) are simply added together, the combined force is not necessarily stable. In our early work we would often get vibration and unstable behavior when touching both the constraint wall and the surface. While a complete treatment of this issue is beyond the scope of this paper, we did find a simple solution.

Our method of guaranteeing haptic stability while combining stable forces in the MarsView user interface is to keep the forces linearly independent whenever possible, and when it is not possible, attenuate them. For instance, since most forces are either horizontal or vertical, it is easy to show that they are linearly independent – in zoom mode, the horizontal "holding" force is independent of the vertical "detent" force. This leaves only two cases in the user interface where forces need to be attenuated. During normal mode, the horizontal wall and surface forces are separated with a linear fade – the surface force fades to zero at a faster rate than the wall force increases. During pan mode, the inertial forces are separated from the wall forces by causing the mass of the surface to go to zero as soon as the wall is contacted. This cuts off all inertial forces and dissipates any surface momentum.

4.3 Data Sources

Our primary data set is a high resolution topographic grid of Mars generated by the Mars Orbital Laser Altimeter project, or MOLA. The MOLA team processes and packages data from a laser range finder orbiting Mars on the Mars Global Surveyor spacecraft and periodically generates an elevation grid (or EGDR) of the entire planet. The resulting data files, free to download on the internet [NASA n. d.], contain up to 66 million data points at 1/32 degree per pixel spacing (approximately 1.9 km).

Of course, any type of elevation data may be rendered with the MarsView algorithm, so we added support for topographic maps of earth that can be obtained from the USGS (U.S. Geological Survey). The USGS offers Digital Elevation Models, or DEMs, with as little as 10 m spacing between each grid point and coverage of the entire United States. Additionally, the USGS GTOPO30 data set offers a series of maps which give global coverage at 30 arc-second resolution (approximately 1 km). Both of these data sets are available free to download off of the Internet [USGS n. d.] or on various media for a small fee.

4.4 Results

Table 2 summarizes the results of MarsView performance tests for each platform listed in Table 1 using three surfaces: MOLA data sets at 1/16 and 1/32 degrees per pixel and the Central America GTOPO30 data set. The high resolution MOLA data set was not tested on the Toshiba laptop because it would not fit into memory. Haptic update time, or the amount of processor time used for haptic computations each millisecond, was measured using the HLOAD.EXE program (distributed with the PHANTOM device drivers). Baseline values were obtained by holding the cursor away

Data Set	Samples/Polygons	Average Haptic Update Time (milliseconds) Toshiba 3000-s353	Dell PWS530
MOLA 1/16 deg/pix	5760x2880/33.2 mil	.13 - .17	.08 - .10
MOLA 1/32 deg/pix	11520x5760/133 mil	n/a	.08 - .12
GTOPO30	4800x6000/57.6 mil	.13 - .21	.08 - .12

Table 2: Test results showing haptic computation time during each update (updates occur every millisecond).

from the surface and maximum values were obtained by swiping the cursor as quickly as possible across the surface.

Table 2 shows that the proxy graph contact model is quite efficient for surfaces consisting of millions of polygons: in all cases, haptic computations take up less than 25% of available processing time. It is also important to notice that the baseline value, which occurs when the user is not touching the surface, contains the overhead of the haptic process (such as drivers) and the actual processing of the proxy graph algorithm is the time above this baseline. Since the running time of the algorithm will grow linearly with the number of triangles that the proxy crosses during each update we could easily get another order of magnitude of resolution in each dimension, or two orders of magnitude more triangles. We extrapolate that a dual 1.7 GHz Pentium III could render a heightfield containing upwards of 10 billion triangles, assuming enough memory was provided (our current implementation would require approximately 50 GB of RAM to store a surface that size using 8 byte doubles).

In order to estimate the frequency of certain proxy graph algorithm events we logged algorithm behavior while exploring the 1/16 degree/pixel MOLA dataset. We found that the algorithm averaged 1.01 collision detection operations per haptic update – the extra 1% beyond the initial collision detection operation is due to the proxy leaving contact with the surface within updates. In addition, when we moved the proxy as fast as possible across the surface the algorithm visited an average of 13.4 vertices during each update – roughly corresponding to 7 to 14 triangles (since no more than two vertices of a single triangle will be visited). This is a significant reduction in the number of collision detection operations since the general proxy algorithm would require at least one collision detection operation per triangle.

In addition to logging collision detection tests and vertices visited we also logged the number of far edge crossings and far edge loops. We found that far edge crossings are very rare on our datasets – no far edge crossings were detected at all in the unscaled dataset. We were able to cause regular far edge crossings by scaling the vertical height of the data by 20 and exploring in steep areas with very narrow, elongated triangles. For instance, we found that zooming in on the walls of *Valles Marineris*, a deep Martian canyon, caused far edge crossings to occur in approximately 3.5% of haptic updates. Far edge loops were even more rare – in the test above they occurred in .1% of far edge crossings. We conclude that far edge crossings are very rare for most topographic datasets.

4.5 Haptic Feel and Texture

While the majority of this paper has been concerned with algorithms, the user experience is arguably the most important feature of a contact model. We found that the proxy graph algorithm results in a haptic rendering that is indistinguishable from other point proxy algorithms. There are no noticable artifacts in surface rendering, including the cases where surface triangles are very large or small due to zooming. Large, smooth areas feel slippery due to the lack of friction while rougher areas tend to feel 'sticky' since the point proxy will get stuck in even the smallest valley in the model.

One advantage of the 'stickiness' of a point proxy is that the texture of the surface is readily apparent to the user – high frequency surface texture is generally felt as a sort of sticky friction. The abrupt change in force when moving from triangle to triangle that is distracting when triangles are very large feels like friction or texture when triangles are extremely small. While an in depth discussion of texture is well beyond the scope of this paper (see [Minsky 1995]) the proxy graph algorithm does enable the user to feel texture due to the high density of triangles.

5 Future Improvements

The primary strength of the proxy graph algorithm is its ability to render very large triangularized models – often the proxy graph algorithm can render much larger models than can fit into system memory. One possibility would be to adapt the proxy graph algorithm to use an implicit data representation such as a subdivision surface or a parametric surface. During the algorithm the local mesh around the proxy could be computed as needed, allowing a detailed surface to be represented very compactly.

Although the proxy graph algorithm can render multiple objects as long as they do not intersect or touch, it would be advantageous to be able to render intersecting objects. One possible avenue of research would be to treat each surface as a constraint and combine them using a method similar to the general proxy algorithm. The main difficulty of this approach would be to efficiently determine if the proxy collides with another surface during constrained minimization.

Some features would be difficult to add to the proxy graph algorithm because of the method of constrained minimization. For instance, since the proxy graph algorithm traces along edges, stable force shading cannot be added by simply perturbing normals as in [Ruspini et al. 1997]. Future research should investigate the addition of force shading and friction by modifying proxy dynamics.

6 Conclusion

MarsView is a successful implementation of the proxy graph algorithm for topographic elevation maps and demonstrates high-performance haptics on standard PC-compatible computers. The height field data structure allows special optimization resulting in successful simulation of surfaces containing more than 100 million triangles. In fact, based on performance results, surfaces an order of magnitude larger could be simulated if they fit in system memory. MarsView also demonstrates that a responsive haptic user interface aids in intuitive exploration of landscape data.

The proxy graph algorithm allows efficient haptic display of very large triangular meshes consisting of hundreds of millions of polygons. Alternation between minimization in free space and minimization constrained by the surface allows the proxy graph algorithm to drastically reduce the number of costly collision detection tests. In addition, by restricting the proxy location to vertices during constrained minimization, the algorithm can use very fast gradient tests to process multiple triangles per haptic update yet preserve algorithm stability. Generally, the small errors that results from tracing along the edges of the triangular mesh are unnoticeable to the user. The main restriction of the proxy graph algorithm is that it cannot handle multiple intersecting objects–this is the focus of our future research.

This work is, in part, motivated by the idea that visual and haptic interaction with information can enable discovery of interesting features and perhaps enable new insights that might not be possible with less complete sensory engagement. By facilitating the simultaneous multi-sensory integration of high dimensionality data we engage our ability for sensory fusion and set the stage for more complete assimilation of the information.

Though the research program reported herein was not tasked to take steps to assess the utility of haptic interaction in planetary science data mining, we have made an important and necessary advance in the technology needed for multi-sensory interaction with such information. By significantly speeding up the rate of haptic rendering, we enable physical exploration of geometric datasets that have *much* greater dynamic range (ratio of largest to smallest physical feature) of information than previously possible. Relevant not only to the current project, we expect this new rendering approach to have value in other haptic technology application domains such as multi-point contact modeling for simulating more realistic and natural physical interactions – such as that required for simulation-based medical training, molecular modeling, and sculpting of digital objects.

7 Acknowledgements

This project was supported, in part, by NASA/Ames Grant NCC-2-1121. We would like to gratefully acknowledge our friends at NASA Headquarters and Ames Research center who helped encourage, launch, and support this project, and especially the efforts of Michael Sims, Butler Hine, Hans Thomas, Ted Blackmon, and Lawrence Edwards at Ames, and David Lavery at NASA Headquarters. In addition, the authors would like to thank Federico Barbagli, Charity Lu, Francois Conti at Stanford for their support and Brian Anthony at MIT for his inspiration and enthusiasm in early project conception.

References

GRANGE, S., CONTI, F., ROUILLER, P., HELMER, P., AND BAUR, C. 2001. The delta haptic device. In *Mecatronics 2001*.

GREGORY, A., LIN, M. C., GOTTSCHALK, S., AND TAYLOR, R. 2000. Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback device. *Computational Geometry 15*, 69–89.

MASSIE, T. H., AND SALISBURY, J. K. 1994. The phantom haptic interface: A device for probing virtual objects. In *Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*.

MINSKY, M. D. R. 1995. *Computational Haptics: The Sandpaper System for Synthesizing Texture for a Force-Feedback Display*. PhD thesis, MIT.

MORGENBESSER, H. B., AND SRINIVASAN, M. A. 1996. Force shading for haptic shape perception. In *Proceedings of the ASME Dynamics Systems and Control Division*, vol. 58.

NASA. Mars orbiter laser altimeter (mola) science investigation. http://ltpwww.gsfc.nasa.gov/tharsis/mola.html.

NESBITT, K. N., ORENSTEIN, B. J., GALLIMORE, R. J., AND MCLAUGHLIN, J. P. 1997. The haptic workbench applied to petroleum 3d seismic interpretation. In *Proceedings of the Second PHANToM Users Group Workshop*.

RUSPINI, D. C., KOLAROV, K., AND KHATIB, O. 1997. The haptic display of complex graphical environments. In *Computer Graphics (SIGGRAPH 97 Conference Proceedings)*, ACM SIGGRAPH, 345–352.

SALISBURY, J. K., AND SRINIVASAN, M. A. 1997. Phantom-based haptic interaction with virtual objects. *IEEE Computer Graphics and Applications 17*, 5 (September-October), 6–10. IEEE Computer Society.

SALISBURY, J. K., AND TARR, C. 1997. Haptic rendering of surfaces defined by implicit functions. In *Proceedings of the ASME Sixth Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 61–68.

STOKER, C., ZBINDEN, E., BLACKMON, T., AND NGUYEN, L., 1999. Visualizing mars using virtual reality: A state of the art mapping tool used on mars pathfinder. Paper presented at the Extraterrestrial Mapping Symposium: Mapping of Mars, July. ISPRS, Caltech, Pasadena, CA.

THOMPSON II, T. V., JOHNSON, D. E., AND COHEN, E. 1997. Direct haptic rendering of sculptured models. In *Symposium on Interactive 3D Graphics*, 167–176.

USGS. Eros data center - products. http://edc.usgs.gov/products/elevation.html.

ZILLES, C. B., AND SALISBURY, J. K. 1995. A constraint-based god-object method for haptic display. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction, and Cooperative Robots*, vol. 3, 146–151.

ZILLES, C. B. 1995. *Haptic Rendering with the Toolhandle Haptic Interface*. Master's thesis, Massachusetts Institute of Technology.