Vol. 3, No. 7, Haptics-e, Jan. 2006
http://www.haptics-e.org

Voxel-Based 6-DOF Haptic Rendering Improvements

William A. McNeely, Kevin D. Puterbaugh, James J. Troy
Boeing Phantom Works
Seattle, WA

ABSTRACT

An approach is presented for realizing an order-of-magnitude improvement in spatial accuracy for voxel-based 6-DOF haptics. It trades constant-time performance for greater spatial accuracy. This helps to make 6-DOF haptics applicable to extraordinarily complex real-world task simulations, which often admit no other known solution short of physical mockup. A reduction of haptic fidelity is tactically incurred but simultaneously mitigated by augmenting standard voxel-sampling methodology with distance fields, temporal coherence, and culling of redundant polyhedral surface interactions. This is applied to large-scale haptic scenarios involving multiple moving objects and to collaborative virtual environments.

Keywords: Haptics, physically based modeling, collision detection, voxel sampling, collaborative virtual environments

1. INTRODUCTION

The voxel sampling approach to 6-DOF haptics demonstrated the potential for simulating real-world tasks such as maintainability assessment of engineering design [14]. In its simplest form, voxel sampling also provides constant-time performance, which directly solves the problem of providing reliable 1000Hz haptic refresh rates without requiring decoupled simulation and haptic loops, which in turn avoids intermediate representations [15]. However, constant-time performance exacts a steep price in spatial accuracy. While some real-world tasks can be simulated adequately using, say, 10~15mm voxels for a scenario with the size and complexity of an automobile engine compartment, many more tasks would become accessible at smaller voxel size such as 1mm or even less. Since the number of surface point samples in the moving objects varies inversely as the square of voxel size, scenarios with realistically sized moving objects and/or small voxel size may easily exceed a million points. However, modern processors can perform only about 2000 point-voxel intersection tests per haptic frame, which falls short of satisfying constant-time performance by more than two orders of magnitude. Spatial accuracy is so important to real-world applicability that alternatives to constant-time performance should be sought.

We found that the spatial accuracy of voxel sampling may be improved by an order of magnitude, still without requiring decoupling the simulation and haptic loops, by sacrificing constant-time performance while introducing a suite of performance-enhancing measures. The most conspicuous cost of this approach is a scenario-dependent viscous-like feeling at the haptic interface, although this is made more acceptable by an enhancement to haptic stability. The performance-enhancing measures presented here include:

- Distance fields (discussed in Section 3)
- "Geometrical awareness," which culls point-voxel samples by reducing surface-contact redundancy that is inherited from the underlying polyhedral representations (Section 4)
- Temporal coherence based on distance fields, dead reckoning, and the voxel tree that underlies the surface point samples of the moving object (Section 5)

The best measure of success of this approach is that it enables the haptic simulation of exceedingly complex tasks that cannot be simulated by any other known means short of physical mockup. This has been demonstrated for a series of real-world engineering tasks in aerospace and automotive applications. A part removal task from one of these environments will be discussed in this paper.

This complex tradeoff is proving acceptable in design-oriented applications such as assessing maintainability and producibility for complex mechanical assemblies, and indeed, it is enabling a new level of functional capability.

In addition to performance-enhancing measures, the paper also discusses some of our findings associated with implementing advanced voxel-based haptics for single and multi-user applications (Section 6) and results of performance testing for complex virtual environments (Section 7).

2. RELATED WORK

Modeling with polygons offers greater spatial accuracy than can be attained using voxels. However, polygon-based 6-DOF haptics is subject to severe performance barriers that limit its practical applicability, for example, by imposing a convexity requirement that constrains scenarios to 10~100 pairs of convex primitives [9]. A single concave object such as a dish antenna or a helical-spiral tube may decompose into hundreds or thousands of convex primitives, depending on modeling accuracy. Polygonal decimation may be used to reduce the number of convex primitives, but at the cost of compromising polygonal accuracy.

NURBS-based modeling offers superior spatial accuracy for 6-DOF haptics, but at the cost of even greater performance barriers and shape constraints. This approach has demonstrated the ability to simulate only relatively low-complexity scenarios that may not contain surface slope discontinuities such as sharp edges [17].

The voxel sampling approach of [14] imposes no formal shape constraints nor complexity constraints on the static objects. In this paper we adopt major elements of that approach, including 3-level 512-tree voxel hierarchy and tangent-plane force model. However, we abandon the goal of constant-time performance, because it limits scenarios to poor spatial accuracy and/or small moving objects, and we compensate by introducing a suite of performance-enhancing measures. The main cost of this change is to incur a tradeoff between the number of points in the moving object and haptic fidelity, but this is acceptable for our purposes, because it enables greater spatial accuracy and/or larger moving objects. The approach of [14] may be extended straightforwardly to support multiple moving objects [21] as well as multi-user collaborative virtual environments, which will be discussed later in this paper. Quasi-static approximation has been introduced into voxel sampling [22], which avoids the need for a contact-stiffness-limiting heuristic, although it sacrifices dynamic realism. In other voxel-based haptics implementations, [20] presents a method for improving the pointshell accuracy and device stability and [19] discusses a method to reduce the number of collision tests for multiple object pairs. Distance and color volumes are discussed in [4].

Geometrical awareness was exploited as core of the state-of-art feature-based collision detection algorithms between polyhedral objects [6,7,13,16]. Geometrical awareness is most often used to compute the distance between two disjoint polyhedra by considering only the two nearest features (vertex, edge, or face) between them. This lends itself to temporal coherence by tracking only the most recent nearest feature pairs. Most implementations impose a convexity requirement, but this is not strictly necessary, and so we avoid it. Another significant difference is our use of voxel-based distance fields to avoid expensive on-the-fly computation of feature-separation distance.

Like voxel sampling, the sensation preserving method of [18] achieves 6-DOF haptic update rates for complex objects by trading accuracy for speed while maintaining similar interaction forces. This method pre-processes the objects by breaking them down into convex pieces and multiple levels-of-detail (LOD). One of the compromises is that its filtered edge collapse technique allows for some object-to-object interpenetration. Pair-wise testing for multi-object collision is also used in this method.

In another 6-DOF method, [11] developed a technique to compute local minimum distances (LMDs) using spatialized normal cone hierarchies. Gradient search techniques have been used to refine the process to achieve haptic rates for moderately sized moving objects. The technique requires a pre-processing step to produce the spatial hierarchy.

Unlike our approach in which voxels replace polygons as representation primitives, voxels may be used to accelerate the search for intersecting polygons in a polygon-based approach [3]. However, any polygon-based approach incurs the performance barrier mentioned earlier, e.g., simulation update rates limited to low hundreds of Hz and moving objects limited to low tens of thousands of polygons. The tactics of loop decoupling and intermediate representations are helpful in this context [15], but they also degrade motion realism and incur the risk of force artifacts.

In the area of collaborative virtual environments, [5] developed a time-delayed collaborative haptics application where one user at a time had control of a single moving object. A shared virtual environment with simultaneous haptic interaction was demonstrated by [12] over long distances. Stable simulation was achieved with 150-200 ms of one way delay, but was limited to point contact interaction. A room-sized simulation trainer with 6-DOF force feedback was built by [10] for astronaut EVA that allowed two crew members to cooperatively manipulate the same large object. Simultaneous interaction with time delay was not taken into account.

2.1 Voxmap PointShell Review

This section is intended to help clarify and review some of the concepts of voxel-based collision detection associated with our earlier Voxmap PointShell ™ (VPS) paper [14].

Figure 1 shows the three basic steps associated with our voxel-based haptics process. The first step is determining which points of an object's pointshell representation have come into contact with the voxelized representation of another object (a). The magnitude and direction of the collision forces are then determined. This process uses a model in which a force vector is calculated based on the depth of penetration along a normal perpendicular to a plane tangent to the object's surface at each point (b). The sum of the collision forces are applied to the dynamic object and numerically integrated. The resulting motion is transferred though a virtual coupling (c) to generate the forces presented to the haptic device and the user. (A multi-user version of this technique will be discussed later in this paper.)

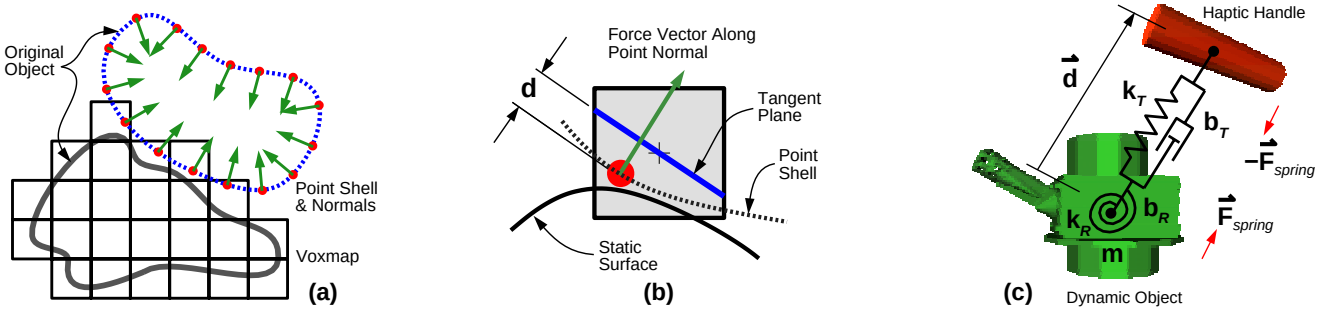

Figure 1. (a) Point-voxel collision detection, (b) Tangent plane force model, (c) Virtual coupling model

VPS works equally well for convex and concave objects, and it does not require decoupling the haptic and simulation loops. It uses a 3-level voxel hierarchy based on a 512-tree, where the leaf node is voxel, the middle level is called “chunk” and contains 512 voxels, and the highest level is called “hyperchunk” and contains 512 chunks.

One of the aspects of this method that needs some clarification is the issue of multiple moving parts. The original VPS paper described the interaction between a pair of objects, where one object is represented as a collection of surface points and the other as a group of voxels. A common misconception is that one of the objects is not allowed to move. This is incorrect — both objects in the collision pair are allowed to move.

All objects are voxelized in their initial positions in the global frame. Now consider the collision of a pair of objects that are both moving. In order to avoid the computationally expensive step of re-voxelizing objects at run time, the positions and velocities of the points of the pointshell object (the smaller of the two, for performance reasons) are transformed into the coordinate system of the voxmap object. The collision is analyzed in that frame, and the resulting force and torque are transformed back into the global frame and applied to the objects in their current positions.

For the simplest case, where only one of the objects is moving, the voxel object’s coordinate system aligns with the world coordinate system and additional coordinate transformations are not required. For simplicity, this was the situation described in the earlier paper, but in the general case where both objects may be moving, the transformation to the voxel object’s coordinate system will be needed.

Also, either object in the collision pair can be a combination of multiple parts that have been merged together to create a single logical object. Merged objects still maintain individual visual attributes, like color, but merged parts behave as a single entity from the collision detection method’s point of view.

Environments with more than one pair of moving parts are also possible. The main issue is maintaining a list of collision pairs. For the worst case, the number of object pairs that need to be tested at each update is:

$$Npairs = n(n-1)/2 \qquad \text{(Eqn. 1)}$$

where n is the number of moving objects.

In the VPS method, each object is treated dynamically as a free object for the duration of a time step, subject to forces of collision with neighboring objects plus any external forces. This is only an approximation, of course, but it becomes a better approximation as the time step decreases, and it asymptotically approaches correct multi-body dynamic behavior in the limit of zero time step.

Other moving object issues associated with multi-user, collaborative virtual environments will be discussed in Section 6.1.

3. DISTANCE FIELDS

It is useful to have advance warning of potential contact between pointshell and voxmap objects. For example, such warning is required by the temporal coherence technique described in Section 5. For that reason we extend the voxelization of an object beyond its surface into free space surrounding the object, marking such free-space voxels with integer values that represent a conservative estimate of distance-to-surface expressed in units of voxel size. This creates a voxel based distance field, as illustrated in the 2D example of Figure 2.

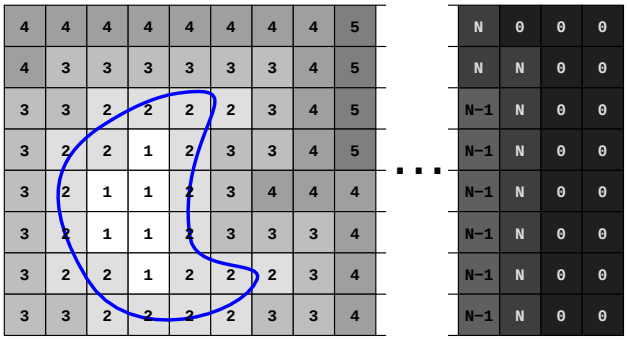

Figure 2. Voxel based distance field (in 2D)

We employ a simple “chess-board” distance-transformation algorithm [2] to calculate the distance field, which gives a conservative estimate of Euclidean distance along non-axis-aligned directions.

VPS supports 2, 4, or 8 bit voxels. The smallest positive value(s) are conventionally reserved for interior voxels, which in Figure 2are marked 1. The distance field extends out to a user-specified maximum value, constrained only by the integer range.

Unless noted otherwise, we assume the use of 4-bit voxels in this paper, since that is a practical choice for haptic applications in current computing environments. For 4-bit voxels the outermost positive voxels could be marked with values up to 15, representing a distance-to-surface of 13 voxels. However, the hierarchical extension of temporal coherence (Section 5.1) works optimally when the maximum distance-to-surface is matched to the power of the voxel hierarchy. Since we use a 512-tree [14], and 512 is the cube power of 8, the optimum maximum distance-to-surface is 8, corresponding to voxels marked 10 (since surface voxels are marked 2). Consequently, values 11 through 15 of the 4-bit range are unused.

The “geometrical awareness” technique described in Section 4 requires three different types of distance field, based on distance to selected geometrical features (vertex, edge, or face). Each field is independently pre-computed and packed into a word. For 4-bit voxels this implies 16-bit words, where the remaining 4 bits are unused. When discussing voxel bitwidth one must be careful to specify whether it refers to the bitwidth of an individual distance field or, less rigorously, to the size of the word required to store all three distance fields. Whenever the expression “16-bit voxels” is used in this paper, it refers to 16-bit words containing three distance fields of 4 bits each.

3.1 Other Proximity-Based Applications

One of the distance field applications that we have developed (and is now part of the VPS API) is a function that colors vertices of the dynamic object model based on its proximity to other objects. Figure 3 shows distance fields used for proximity-based coloring (warmer colors indicate closer proximity). The main ben-

efit from proximity coloring is that it aids haptic interaction by visually conveying distance to contact.

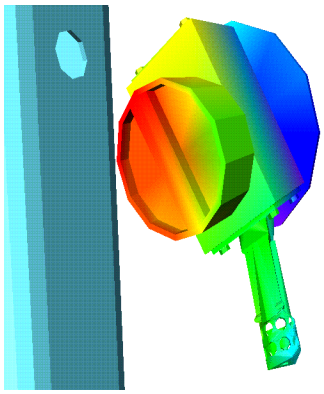

Figure 3. Voxel-based proximity coloring

Applications that use static environment voxel data are also possible. Highlighting surface voxels within a specific distance to the moving object produces a shadow-like effect that can also aid in distance-to-contact perception. Proximity-based distance measurement can be used to give a reasonable approximation for quickly determining minimum distances. The distance gradient information could also be useful for path planning, similar to potential field based path planning applications.

3.2 Collision Offsetting

Forces are generated using the tangent plane model, as reviewed in Section 2.1. One is free to select the voxel layer in which tangent-plane forces are generated, which we refer to here as the "force layer." If one selects surface voxels for that purpose, then as a potentially undesirable side effect, the exact surface of the pointshell object (e.g., its polygonal representation) may, in general, interpenetrate the exact surface of the voxmap object. In order to minimally avoid exact-surface interpenetration, one must adopt the second layer of free-space voxels as the force layer [14]. Since the distance field extends farther than two layers into free space, one may move the force layer to even more distant free-space layers and thereby create a collision offsetting effect. This is useful in task simulations where additional clearance is needed but is not formally modeled, e.g., to allow for human grasp in a part-manipulation task. In VPS one can dynamically vary the force layer and thereby dynamically vary the amount of clearance.

One might consider extending this scheme to the pointshell. The pointshell is normally derived from the centerpoints of surface voxels, but a free-space voxel layer might also be used for that purpose. However, free-space layers contain more voxels than the surface layer, and VPS performance degrades as pointshell size increases. For that reason, VPS derives the pointshell from the surface layer, except in the situation that the user requests a range of collision offsetting that exceeds what is achievable by dynamically varying the force layer inside the voxmap object. In that case, VPS derives the pointshell from the free-space layer that is both nearest the surface and minimally satisfies the user's requested range of collision offsetting.

Despite the static nature of the pointshell as described above, it is possible to dynamically vary the locations of the points in the pointshell, by displacing them a short distance along the direction of the surface normal, either toward free space or toward the interior of the object. This provides the capability of fine-tuning the amount of collision offsetting. However, this has the problem that, depending on the direction of displacement and the local curvature of the surface, the displaced points may spread apart, creating a looser mesh of points that runs the risk of undetected penetration. One way to counteract this effect is to select a voxel size for the pointshell object that is smaller than that of the voxmap object, at the price of tactically degrading VPS performance.

An interesting application of pointshell displacement is mating-surface simulation, as illustrated in Results (Section 7) for a simple ball-and-socket scenario. In general, mating-surface simulation is problematic at haptic speeds, in the absence of kinematical constraints or similar special-case information, because manifold surface contact is computationally expensive. If mating parts are permanently constrained within a mechanism for the entire duration of a motion scenario, then kinematical constraints are certainly appropriate. However, it becomes problematic when kinematical constraints may engage or disengage during a simulation. For example, if a wrench can be used on a certain type of fastener, then the simulating system must know that association in advance. Any subsequent changes to tool or part geometry are liable to invalidate that association. Furthermore, the simulating system must somehow decide when to engage the constraint and when to disengage it, e.g., by detecting that the tool is sufficiently aligned with the part to engage the kinematical constraint. This leads to artifacts such as a mysterious attractive force that acts to seat the tool whenever is it sufficiently aligned with the part. Another artifact is a sticky feeling when trying to disengage the tool. VPS suggests an approach, albeit a computationally expensive one, to avoid such problems and artifacts by avoiding kinematic constraints altogether.[1]

4. GEOMETRICAL AWARENESS

Although the approach presented here is voxel-based, voxels may inherit properties of their parent polyhedral objects at discretization time, which has great value in culling point-voxel intersections at run time, as explained below.

To begin, consider the interaction of a pair of rigid non-penetrating polyhedral objects. Consider their surfaces as a pair of point manifolds that exhibit an arbitrary (even infinite) number of point intersections (surface contacts) for a given configuration. For physically-based modeling purposes, the only interesting contacts are those where one or both points belong to a C1 discontinuity in their respective parent surface. As a simple 2D example, the only interesting contacts between two blocks are their vertex-edge contacts, as illustrated in Figure 4.

In 3D, only vertex-surface and edge-edge contacts are interesting. ("Surface" is understood to include its edge boundaries and "edge" its vertex boundaries, hence edge-vertex and vertex-vertex

1. Developers are still free to create additional constraints on top of the basic VPS collision detection implementation.

contacts are both trivial subsets of edge-edge.) We refer to this powerful insight as geometrical awareness, to adopt the terminology of [6]. This result is entirely general for non-penetrating polyhedral objects, in particular, it does not require convexity. One may ignore all surface-surface and surface-edge contacts, which effectively reduces the problem's dimensionality and reduces computational load enormously.

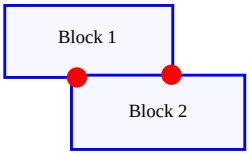

Figure 4. One 2D block rests upon another 2D block. (Red circles represent vertex-edge contacts.)

Geometrical awareness can be applied to voxel sampling as follows. Point samples are taken as the center points of surface voxels. One labels each point as vertex, edge, or surface, according to whether its parent voxel inherited as a "priority feature" the vertex, edge, or surface attribute, respectively, from the underlying polyhedral geometry. By "priority feature" we mean the following priority ordering of feature inheritance. If a point's parent voxel intersects (i.e., contains) one or more vertices in the polyhedral geometry, then the point is labeled as vertex, even if its voxel also intersects edge or surface elements. Similarly, an edge point's voxel intersects one or more edges but no vertex, while a surface point's voxel intersects one or more surfaces but neither edge nor vertex.

To more efficiently apply geometrical awareness to point-voxel interactions such as in the tangent-plane force model, we precompute three voxel-based distance fields toward the nearest surface-, edge-, and vertex-voxel, respectively, as described below. Thus, one uses surface points to sample the vertex-distance field, vertex points to sample the surface-distance field, and edge points to sample the edge-distance field. Figure 5 shows edge and vertex distance fields for an arbitrarily shaped polygonal object.

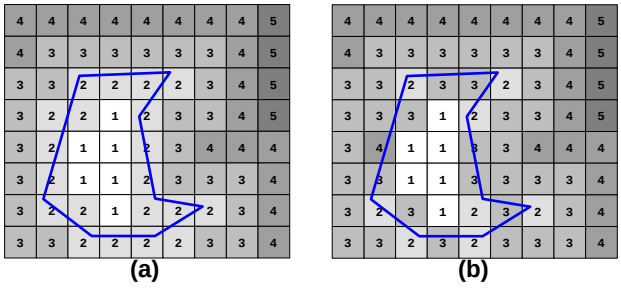

Figure 5. Edge (a) and vertex (b) distances fields.

A known limitation of geometrical awareness is that it is not effective against manifold contact of 3D edges (e.g., a sword's edge perfectly aligned with another sword's edge). In that case, geometrical awareness prescribes testing a potentially large number of point-voxel contacts along the linear region of overlap. It is not clear how to generalize geometrical awareness so as to address both the common form of edge-edge contact (e.g., swords crossed at an angle) and the exotic case of edge-edge congruency. Fortunately, the latter almost never occurs in practical scenarios, not even within the accuracy of a voxel size.

4.1 Optimizing Voxel/Polygonal Accuracy

Feature-based distance fields are most effective when the accuracy of the underlying polyhedral geometry matches voxel accuracy, for the following reason. As one increases polyhedral accuracy (holding voxel size constant), one obtains more polygons of smaller dimensions, which increases the likelihood that a given voxel will contain a vertex and/or an edge. That increases the number of vertex-surface and edge-edge interactions at the expense of surface-surface interactions, which tends to defeat geometrical awareness and degrade performance. To compound matters, polyhedral accuracy is typically much better than voxel accuracy. Often it is decided casually, e.g. in the process of exporting it from a CAD system, oblivious to voxel size.

For best results, therefore, polyhedral accuracy must be reduced to voxel accuracy. We accomplish this through a process similar to decimation[2], at voxelization time, as follows. First, tessellate the polyhedral facets into triangles. Then, if any pair of adjacent triangles has the property that its non-shared vertices deviate from coplanarity by less than 1/2 voxel size, and also if their polyhedral angle is less than 90 degrees, then that pair of triangles is treated as a single quasi-planar quadrilateral for voxelization purposes. Otherwise, if those criteria are not met, then the pair of triangles remains dissociated. This process is repeated by considering triangles adjacent to a quasi-planar quadrilateral, which may lead to a quasi-planar pentagon, etc. After all triangles have been so processed, distance fields are constructed from the features of the resulting quasi-planar polygons. The 90-degree polyhedral-angle criterion prevents small curved objects (such as a sphere with diameter less than a voxel size) from being reduced to a single planar polygon.

5. TEMPORAL COHERENCE

The voxel sampling method provides a natural opportunity for exploiting spatial and temporal coherence, or temporal coherence in short. This is done by tracking and predicting the status of points in the "pointshell" (set of surface point samples) of the dynamic object. A point that contacted a surface voxel in the previous frame is likely to remain in contact in the current frame.

Whenever a point samples its appropriate voxel-based distance field (see Section 3), it obtains a conservative estimate of its minimum distance from any contact. If we also know the point's maximum speed, then by dead reckoning we can predict how many frames will elapse before contact can possibly occur, which allows us to safely reduce the frequency of point sampling. Hence, the pointshell may contain more points than could possibly all be tested in a single haptic frame, and since the pointshell is derived from surface voxels, this enables the use of smaller voxels and greater spatial accuracy.

This requires knowing a point's maximum speed, but the latter is formally unlimited. A more serious problem is that the known

2. Note that polygon decimation does not change an object's voxelization.

speed may be so large that the available processing power cannot keep up with the burden of predicting contact for all free-space points. To solve these problems, we impose a speed limit that applies to all points. For this purpose we denote the maximum distance that any point may travel in a haptic frame as *MaxTravel*. In general, *MaxTravel* is adjusted on a frame-by-frame basis because it varies inversely with the number of points that require testing during that frame. As the amount of contact and near-contact increases more point tests become necessary. It is mandatory to test points that were in contact in the previous haptic frame. However, free-space points may be scheduled for testing at a reduced frequency.

MaxTravel has an absolute upper bound of 1/2 voxel size[3], in order to prevent points from skipping over the penalty-force region of surface voxels and penetrating into the object's interior. Since the time duration of haptic frames is constant, *MaxTravel* is equivalent to a speed constraint. This expresses itself at the haptic interface as a viscous-like resistance whenever the virtual-world speed tries to exceed *MaxTravel* per haptic frame. For example, consider a scenario modeled using 2mm voxels and 1000Hz haptic refresh rate. A maximum speed of 1/2 voxel per millisecond is 1 meter/second. This corresponds to user motion of roughly one arm's length per second, which is unrealistically fast in the context of any application that involves manipulating objects with careful intent. In this simple example, therefore, the *MaxTravel* constraint has negligible impact at the haptic interface. However, in a more complete analysis (1) the speed constraint applies to every point on the object's surface, which generates a more complicated constraint on the object's overall translational and rotational velocities, (2) any spatial scaling between the virtual world and the haptic interface must be considered, (3) *MaxTravel* may be smaller than its absolute upper bound of 1/2 voxel, as calculated below:

$$MaxTravel = \frac{nCapacity - nMandatory}{\sum \frac{n_i}{0.5s \cdot i}} \qquad \text{(Eqn. 2)}$$

where *nCapacity* is the number of point tests that the processor can perform per haptic frame, *nMandatory*, is the number of "mandatory" tests (for points already in contact), n_i is the number of points in free space at i voxels (i>0) from contact, and s is voxel size. If Equation 2 yields *MaxTravel*<0.5s, then we limit *MaxTravel* to its absolute upper bound of 0.5s. The value of 0.5s is used here because it represents the minimum possible distance from the center of a voxel to a neighboring voxel.

The worst case is that of more mandatory tests than can be performed, in which case *MaxTravel* in Equation 2 becomes zero or negative and further motion becomes impossible. Whenever this happens, VPS is unable to meet the user-requested time constraint, in which case it tests all mandatory points and abandons any attempt to maintain time criticality. However, in practice, geometrical awareness (Section 3) so sharply reduces the number of points in contact that we have rarely encountered this worst-case situation during a series of complex real-world task simulations.

We track and update point status using a mechanism called distance-to-contact queues. All points that currently have the same distance-to-contact value are considered to belong to the same value-specific queue. However, those points beyond the range of the distance fields belong to the same queue as those lying at a distance of exactly one voxel beyond that range. Therefore, n_i in Equation 2 is the number of points in queue i. (Since we use 4-bit distance fields, the number of queues is 16.) In general, there will not be enough processing power to test the entire contents of each queue during the current haptic frame, but it is only necessary to test the entire contents of the mandatory-point queues plus the following number of points m_i of each free-space queue i:

$$m_i = MaxTravel \cdot \frac{n_i}{0.5s \cdot i} \qquad \text{(Eqn. 3)}$$

where m_i is rounded up to the nearest integer. We test m_i points per frame in round-robin fashion for each queue individually. This ensures that no point may travel into penetration undetected, i.e., before being re-tested. Whenever a point is re-tested, its distance-to-contact value may change, which then causes the point to migrate to a different queue. We make the assumption, borne out by observation, that *MaxTravel* varies so slowly with time that it may be considered constant while a point is waiting for re-testing. In fact, *MaxTravel* tends to be conservative, because its value typically decreases with time whenever objects are approaching contact.

The distance-to-contact queues are implemented as follows. Each queue is a bitmapped representation of the entire pointshell. Each point is represented as a one bit in just one of the queues, and for all other queues the bit at this same address is zero. During each haptic frame, a fraction of each queue's contents is traversed in order to satisfy the minimum sampling frequency. Whenever a one bit is encountered, its associated point is sampled.

Under this implementation, distance-to-contact queues become quite sparse. To accelerate their traversal, each queue is ordered into a 2-level hierarchy. The leaf level contains individual bits of the queue, while the upper level contains bits that are one whenever any of its 32 leaf-level children are one. This enables the skipping of entire 32-bit runs of zero bits. When n_i is zero, the entire queue is empty and may be skipped. While it may not be obvious that this implementation is preferable to more sophisticated point-scheduling schemes that can be imagined, in fact it yielded higher performance than several alternatives that we explored.

Temporal coherence conveys an important, if unexpected, benefit for haptic stability. Under virtual coupling (Figure 1c), the most likely source of instability is large transient movements of the dynamic object. However, *MaxTravel* inherently prevents large transient movements. Stability is a very complex topic, and there are many other possible sources of instability (e.g., limit-cycle oscillations, overly stiff virtual systems, unpredictable user-applied forces, device limitations, etc.). However, empirically, the stability benefit from *MaxTravel* has enabled perfectly stable haptic operation for all scenarios that we have ever tested.

3. If the objects are sufficiently far apart, this upper bound may increase to 1/2 hyperchunk size, as a consequence of Hierarchical Temporal Coherence (Section 5.1)

5.1 Hierarchical Temporal Coherence

Since the pointshell is derived from the centroids of surface voxels, it inherits the spatial hierarchy of its parent voxel tree. All points that came from the same "chunk" of the voxel tree (the first level above leaf level) are assigned to contiguous bit addresses in the distance-to-contact queues. Then, whenever the entire chunk's worth of points is known to lie in free space, we may remove all such points from their queues and continue tracking only the chunk's distance-to-contact, e.g., by testing the chunk's centroid against the surface distance field. (Since the chunk's contents may be marked with a mixture of surface, edge, and vertex attributes, we must test against the most conservative distance field, which is the surface distance field.) This greatly reduces the point-testing burden, since in a 512-tree, a chunk contains about 100 points on average.

One may learn whether a chunk's entire point contents lie in free space as follows. Chunks are marked with a discretized distance-to-contact value in the same manner as voxels, thereby creating a chunk-level distance field. The pointshell-object's chunk centroid is then used to sample the static-object's chunk-level distance field, in precisely the same manner as point-voxel sampling. If such a test reveals that a chunk lies beyond the space spanned by voxel-level distance fields, then that chunk is considered to lie entirely in free space, and chunk-level temporal coherence is applied. On the other hand, if a previously free-space chunk enters the space spanned by voxel-level distance fields, then its contents are disgorged and re-inserted into the point queues. (The cost of such transitions may be greatly reduced by exploiting the fact that the points have contiguous bit addresses.)

Point sampling and chunk-centroid sampling behave identically in all respects except the following. "Contact" is re-defined to mean that the chunk enters the space spanned by voxel-level distance fields, as described above. Every chunk that lies in that space is considered to occupy a mandatory chunk queue. *MaxTravel* is modified straightforwardly in Equation 2 by augmenting *nMandatory* with a chunk-specific contribution and also extending the summation over queues to include the new chunk-level queues.

5.2 Point Drifting

As a significant performance optimization, one may reduce the frequency of voxmap lookup during point testing, as follows. Whenever voxmap lookup becomes necessary (as explained below), the point's current exact spatial position is stored, along with its current voxel-accurate distance-to-contact (as discovered through voxmap lookup and expressed implicitly by the point's distance-to-contact queue number). Subsequently, whenever that point falls due for testing under temporal coherence, one first computes its point drift, defined as the exact distance between its current position and its previously stored position. If so much drift has occurred that the point may be "too near contact" (as defined below), then voxmap lookup becomes necessary and drifting begins anew. Otherwise, if the amount of drift is not so great, then voxmap lookup is avoided, and the point is allowed to continue drifting. The criterion for being "too near contact" is that the point could possibly have drifted as much as two queues away from contact. In principle, one could more aggressively wait until it was only one queue from contact, but we elect to have a one-queue margin of safety.

When a point begins drifting, it stays in its initial distance-to-contact queue until the amount of drift is more than a voxel size. Whenever re-queueing becomes necessary, we conservatively assume that the point moved nearer to contact, i.e., to a lower-numbered queue. That incrementally increases the frequency of testing, but empirically, each test suddenly becomes about 7 times faster by avoiding voxmap lookup. This 7-fold advantage decreases as drifting proceeds, becoming minimal when the point drifts as near as two queues from contact, but when that happens the point is re-tested subject to voxmap lookup and properly re-queued, and drifting begins anew. The net performance benefit of point drifting depends in a complicated way on the motion scenario, but typically it is several-fold.

5.3 Dynamic Pre-Fetching of Voxel Data

It may easily happen that there is insufficient system memory to hold all voxel data for a given scenario, especially for large-scale scenarios and/or small voxel sizes. Under 32-bit operating systems the addressing limit is 4GB, which is often reduced further to 3GB or 2GB. While virtual memory is a good solution for non-time-critical applications, it is fundamentally incompatible with time-critical haptics. Just-in-time memory paging causes highly distracting force discontinuities or even haptic-controller timeouts. To avoid such adverse effects, one needs a predictive memory-paging scheme. For that reason, we implemented a dual-thread scheme that supports time-critical operation at haptic rates in one thread, coupled with dynamic pre-fetching of voxel data in the other thread.

A convenient way to implement dynamic pre-fetching is to integrate it with chunk-level temporal coherence as described in Section 5.1. The latter includes probing the space that lies beyond the space spanned by voxel-bearing chunks in the static distance fields. Consequently, one can readily detect when a given chunk of the dynamic object has reached a distance of one chunk size away from any voxel-bearing chunk(s) in the static distance fields. Whenever that happens (to recall Section 5.1), one immediately switches level-of-detail representations in the dynamic object, from using the chunk's centroid to using its constituent points. To extend that mechanism to dynamic pre-fetching, simply treat such representation-switching events as requests that voxel-bearing chunk(s) of the static distance fields should be fetched into real memory, if necessary. A separate thread can then perform such fetching in time to satisfy access by the haptic thread.

There is no way to guarantee that a pre-fetching thread can always act fast enough to satisfy the haptics thread, depending on the speed of the hard drives, scenario complexity, the backlog of pre-fetching requests, size of *MaxTravel* compared to chunk size, etc. To cover all such contingencies, we allow the haptics thread to be temporarily suspended as needed to allow the pre-fetching thread to catch up. During a state of suspension, *MaxTravel* is set to zero, and no forces are sent to the haptic interface. We limit the duration of any suspension to 2 seconds, after which the simulation is terminated. Empirically, even with our largest scenarios, such suspensions occur so rarely and/or have such brief duration that they proved imperceptible. Furthermore, we have not yet

encountered a scenario that was prematurely terminated by the 2-second timeout.

We did not extend this mechanism to hyperchunks, nor was temporal coherence extended to hyperchunks, on the grounds that the complexity of such an extension seemed to outweigh its potential benefits.

6. IMPLEMENTATION

The initial version of VPS [14] has been used at Boeing for several years to analyze many types of complex, real-world engineering problems, but it was limited by the total number of points allowed in moving objects. The enhancements described in the previous sections have been implemented in prototype applications that now allow the analysis of even more complex systems with increased accuracy. We will begin by describing our haptics development environment.

We have used a variety of architectures for experimentation and prototyping, using either one or two computers. For production use we implement the VPS collision detection and force generation algorithms in a separate computer we call the "Haptic Controller" using a client-server model. Our choice for this approach was driven by the mismatch between the computing requirements of physically based modeling and the available workstations used by typical engineering departments. Physically based modeling has these characteristics:

- Computationally intensive — dual CPUs are best so one can be devoted to haptics and the other to secondary tasks
- Large amounts of memory (RAM) are required
- Large amounts of available high speed disk space are needed to save voxel data

Production workstations generally have these characteristics:

- A single CPU
- Modest amounts of memory
- Computation is already taxed by graphical rendering
- Memory fills with data representations optimized for graphical rendering
- Local disk space may be lower speed or inaccessible
- OS and application software installation tightly controlled by IT department

The mismatch between requirements and existing hardware is solved by putting the haptic process on a PC that is devoted to haptic processing. The haptic device (or other type of input device) is then connected to this PC as shown in Figure 6. The Haptic Controller PC is connected to the client workstation via ethernet and TCP/IP. If the PC is given an IP address on the same subnet as the workstation, connecting them via a switch minimizes bandwidth contention and allows them to communicate at 100Mbit/second, regardless of the network connection speed available to the workstation (often much slower). The Haptic Controller PC has no visual interaction with the user, and need not have an associated monitor. The Haptic Controller supports a variety of interaction devices including: various models of the PHANTOM haptic device, 6-DOF Spaceball (and similar) devices with no force feedback, and a 2-DOF mouse with no force feedback.

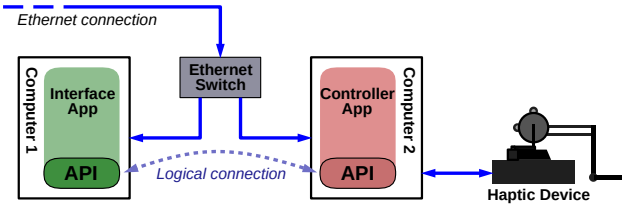

Figure 6. Haptic Controller configuration.

Within the Haptic Controller, one thread is devoted to collision detection and force generation, and a second thread handles communication tasks with the client, and pre-processing. When the Spaceball is used, a third thread receives updates from it.

The Haptic Controller provides these services to the client: voxelization, transparently caches voxel data for later reuse, manages the haptic device, and supplies updated positions of the goal and moving objects on demand. An API is supplied for use by the host application. The API is designed to minimize the intrusion into the host application.

We have used the Haptic Controller with two applications: FlyThru®, a Boeing-proprietary visualization system used for design reviews, and a prototype application used for investigating collaborative haptics. The results reported here were obtained with FlyThru. FlyThru was designed to handle large amounts of geometry and includes rendering optimization for the special case of a fixed eye point with a small amount of moving geometry. This optimization is important because it allows the environment to be rendered in full detail during haptic interaction at responsive frame rates.

High-speed hard drives are desirable for the Haptic Controller for the sake of dynamically pre-fetching voxel data (Section 5.3). Empirically, hard drives with higher data transfer rates (like 10K-15K RPM SCSI drives) are more likely to meet pre-fetching demands for large-scale scenarios. If lower-speed hard drives are used, then haptic force quality acquires a rough and viscous feeling whenever two objects make contact for the first time, due to the fact that *MaxTravel* is set to zero while waiting for voxel data to appear in memory.

6.1 VPS-Based Collaborative Virtual Environments

In addition to building VPS-based applications with multiple constrained and unconstrained moving objects, we have recently implemented a multi-user environment for collaborative 6-DOF haptics that uses VPS for collision detection and response. The types of haptically enabled collaboration that we have been investigating include: design reviews, maintenance access, and training.

Implementing a collaborative virtual environment (CVE) with multiple simultaneous haptic users becomes more difficult when users are located at geographically separate sites. Haptic interaction is very sensitive to synchronization delays produced by communication over large distances. In order to maintain haptic stability, while minimizing the impact on interactive performance, the application needs to be designed with time delay compensa-

tion in mind. In our CVE implementation, we address the delay issue by using peer-to-peer communication and a multi-user virtual coupling configuration. Figure 7 shows our collaborative virtual environment application for maintenance access analysis.

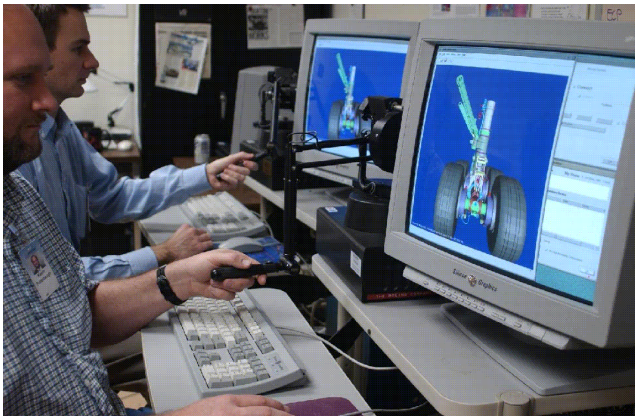

Figure 7. Haptic enabled collaborative virtual environment

The peer-to-peer architecture synchronizes the CVE without a central server[4]. Each user is running a separate simulation of the environment in which models and motions are synchronized with the other users. The implementation uses TCP packets between the front-end graphical interface and UDP packets between haptic controllers. The system supports active users with haptics and non-haptic devices, as well as passive (visual only) users. A user can enter and leave the simulation environment at any time without impacting the other users.

The two main types of collaborative tasks that we have focused on are those involving: (1) each user controlling separate objects, and (2) multiple users controlling the same object. We will refer to these as type-1 and type-2, respectively. Both have the same type of infrastructure with respect to data and model synchronization, network connections, and device control. There are some significant differences as well.

The first type (control of different objects) has the same pairwise collision checking requirements discussed in Section 2.1, but with the added requirement that users be aware that a voxel size mismatch between users will produce an asymmetric force response. A user with a smaller voxel size than other users will create an imbalance in contact forces between objects. This allows user A's pointshell object to contact user B's voxmap and generate repulsive forces before B's pointshell object makes contact with A's voxmap. This gives the user with the smaller voxels an enhanced ability to push/pull other users around without being affected equally by their interactions. Although the exact nature of this imbalance is probably unique to voxel-based haptics, this type of condition is a common problem in collaborative systems without centralized management — for example, in a multi-player video game users can cheat by modifying the local front-end interface to give themselves special powers. In general, collaborative haptics applications will have asymmetric behavior if force calculation parameters are not the same for all users.

The second type of collaboration (users controlling the same object) requires a new type of coupling connection. In previous implementations [14], we have used virtual coupling elements [1][8] to connect the dynamic object to the haptic device. For the multi-user case, the virtual coupling model was extended to connect the instances of the object that all users control. Since each user is running an independent simulation, there is an instance of the object independently calculated for each simulation. Coupling effects from the other instances of the object act as additional external forces on the local dynamic simulation of each object instance. Figure 8 shows this connection for a two user arrangement.

The multi-user virtual coupling effectively creates an environment for bilateral teleoperation of multiple haptic (or robotic) devices, with the addition of collision detection and response from objects and constraints in a virtual environment. One of the interaction drawbacks of this method is the potential for divergence of the multiple object instances. This can occur when another object (like a thin wall) gets trapped between the instances of the dynamic object.

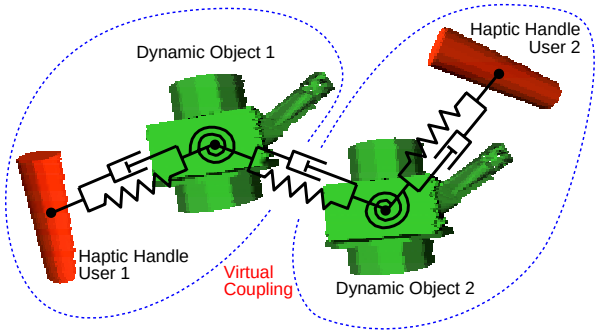

Figure 8. Multi-user connection model using virtual coupling elements

Another interesting finding for both of these approaches to collaboration is that the haptic devices and dynamics simulations remain stable when force information from the other users is transmitted at rates below 1000Hz. The systems were functionally stable when external force updates from the other users were received at 100Hz. Note, we still maintained each users local simulation at 1000Hz to keep numerical integration and haptic loops stable.

A combined environment that simultaneously allows both types of interaction presents some interesting response possibilities. For example, what happens when two users are controlling one object (type-2) and then a third user joins the environment and controls another object (type-1)? In addition to feeling bilateral forces from each other, the first two users will see and feel contact interaction with the third as expected with type-1 contact. From the third user's point of view, he or she will see and interact with what appears to be a single instance of a moving object — unless the users controlling that object enter into a divergent condition. One option for dealing with this situation is to allow user 3 to see and

4. The *collaborative architecture* is peer-to-peer and should not be confused with the *Haptic Controller architecture*, which uses a client server model.

interact with both instances of that object. How well this works from a usability standpoint is still unknown, since a combined environment is not something we have tested yet.

In addition to multi-user interaction issues, time delay compensation is another major concern in collaborative virtual environments. Time delay is especially problematic when users are located at geographically separate sites. It is less critical for the type-1 collaboration, where the non-coupled users may not be aware of the delay — at least not initially. They will still see the other users objects moving and instantly feel forces when they make contact with those objects. The becomes apparent when objects have continuous contact. Although local contact forces are felt immediately, the reaction of the other user's object to the contact is delayed. A similar delayed reaction occurs when the contact is removed. Fortunately, this delay does not appear to destabilize the simulations. But that is not the case for type-2 collaboration.

When multiple users simultaneously control the same object, time delay can cause the haptic devices to become unstable. For this situation, we have implemented a method for linking the current value of the time delay to the stiffness gains in the cross-user virtual coupling. A linear reduction of the stiffness for delays up to 1 second appears to keep both simulations stable. We are currently using gains that have been determined experimentally, and hope to develop a more theoretical basics for gain selection in the future.

6.2 Physically Based Modeling Without Force Feedback

Although we have found that task performance of force feedback applications is superior to physically-based applications without force feedback, the cost of haptic devices appears to be a barrier to widespread adoption in the engineering community. Devices that have 6-DOF input, but no force feedback, like the Spaceball® provide a low cost alternative. Just as with a haptic device, the VPS algorithms prevent part interpenetration and impose physically possible motion. The Spaceball is a force input device where the user pushes/pulls against internal spring-like elements. This motion is then converted into position and orientation data. Other 6-DOF, non-force feedback devices like the MicroScribe® have also been successfully tested with VPS-based virtual environments.

Our informal testing results indicate that these types of devices are adequate for tasks of low to moderate difficulty. We have found that very difficult tasks, like part extraction from congested environments, are only solvable with force feedback. In general, task performance is usually slower without force feedback, but is a reasonable alternative for some conditions. In our implementations, we usually attach the Spaceball to the Haptic Controller PC rather than the host workstation so that any existing use of another Spaceball by the host application (i.e., for view control) is unaffected. With such a system, a sufficiently ambidextrous user can simultaneously change the viewpoint and manipulate objects.

7. EXPERIMENTAL RESULTS

Our high-performance haptic rendering system has been implemented on Linux®, Microsoft Windows®, and SGI IRIX®. The performance results in the following discussion were obtained using a two processor 2.8 GHz Xeon PC with 2GB of RAM running Windows XP. Haptic rendering is performed on one processor to provide updated force and torque information to the haptic device and read position and orientation of the haptic handle in a closed-loop control system running at 1000Hz. Force feedback is provided by a PHANTOM® Premium 1.5/6-DOF haptic interface made by SensAble Technologies, Inc. The host graphics application for these experiments is FlyThru (our internal high-performance visualization system), which uses a second computer to maintain a graphics frame rate of 10~20 Hz, depending on the complexity of the moving geometry, but independent of the amount of static geometry. This high performance is achieved by rendering the static geometry once to the color and Z-buffers, then reusing those images for subsequent frames [23]. (Other graphics display environments are also possible.)

VPS provides the capability to crudely simulate mating-surface scenarios without using kinematic constraints, as described in Section 3.2. This is illustrated here for the simple scenario of a ball that may be rotated in a cradle-like socket (Figure 9a). This example illustrates a worse case scenario where a large amount of object-to-object contact occurs. In this case, the ball is the point-shell object, and its points are displaced by half a voxel toward the interior of the ball, in order to allow the ball to seat fully with the socket. For this scenario we measure VPS performance in terms of the time required for a full rotation of the ball. With a radius of 25 mm and a voxel size of 0.35 mm, this takes 1.28 seconds on a 2.8GHz processor. The speed of rotation is limited by *MaxTravel*, which is determined by voxel size and processor speed. In this scenario there are, on average, 250 points in contact at all times.

Figure 9b shows the 777 Main Landing Gear used here as an example of a large dataset for maintenance analysis tasks. The overall dimensions of this dataset are approximately 4.1 x 1.9 x 4.8 m. The dynamic object chosen for testing is a large hydraulic actuator near the bottom of the scene that measures 0.9 x 0.2 x 0.2 m. For this test scenario, the user interacts with the environment by removing the dynamic object from its installed position. Simulation accuracy was adjusted over multiple tests by varying the voxel size.

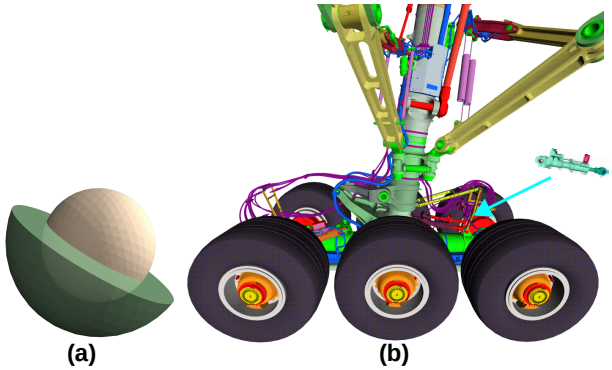

Figure 9. Models used for testing: (a) Ball and socket model, (b) 777 Main landing gear (with dynamic object)

TABLE 1. Virtual Scenario Measurements[a]

Scenario	Voxel Size, mm	Voxelization Time, sec.	Loading Time, sec.	Dynamic Object		Static Environment	
				Triangles	Points	Triangles	Voxels
Ball and socket	0.35	5.8	1.7	2048	23960	2176	5.91×10^5
Ball and socket	0.15	21.5	7.0	2048	130688	2176	3.11×10^6
Landing gear	1.0 / 2.5	1353	333	40476	528653	2.76×10^6	4.59×10^8
Landing gear	0.7 / 1.25	5861	1355	40476	1.14×10^6	2.76×10^6	1.78×10^9

a. The total number of voxels shown in Table 1 includes internal, surface, and distance field voxels (as described in Section 3). When multiple voxel sizes are listed for a scenario, the smaller number is the point spacing of the dynamic object and the larger number is the voxel size of the static environment.

Table 1 collects the parameters of the dynamic objects and the static environments in each of the above two scenarios, in which our goal was able to maintain a 1000Hz haptic refresh rate. Each scenario was evaluated twice, once with a relatively large voxel size and once with a small voxel size in relation to the overall dimensions of the scene. The table includes the sampling resolution (voxel size), numbers of triangles, number of sampling points in each dynamic object, numbers of triangles, and number of voxels in each static environment.

Figure 10 shows additional voxelization data for the landing gear model in Table 1. From this data we can determine that voxelization time is inversely proportional to the square of the voxel size ($t \propto 1/s^2$).

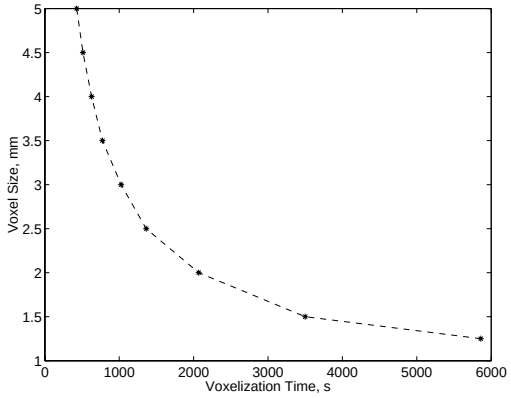

Figure 10. Voxelization time comparison

One cannot straightforwardly assess the relative performance benefits of geometrical awareness and temporal coherence, since they depend sensitively on the motion scenario. However, one may directly compare the currently attainable accuracy (as represented by voxel size) against what was attainable before the advent of algorithmic enhancements such as geometrical awareness and temporal coherence. The maximum number of points that VPS could process in 1999 was reported as 600 [14]. Currently there is no formal limit, but up to 1M points is readily attainable and usable. We must also account for the fact that CPU speeds have increased about 8-fold since 1999. Consequently, 1M points was equivalent to 125,000 points in 1999, a 200-fold increase. Since the number of points varies inversely as the square of voxel size, a 200-fold increase in pointshell capacity corresponds to a 14-fold improvement in accuracy due to VPS algorithmic enhancements alone. Combining this with the CPU-speed increase, there has been a net 40-fold improvement in accuracy since 1999.

Throughout testing, we paid particular attention to motion behavior and quality of force and torque feedback. Artificial viscosity caused by *MaxTravel* (Section 5) was evident, especially at smaller voxel sizes, whenever objects were in contact or nearly so. However, both force and torque feedback are distinctly helpful to performing task simulations.

These results are from experiments performed in a single user environment, but the performance should be nearly identical in the multi-user environment described above, since each user will be running an identical simulation (with a small amount of communications related overhead).

8. SUMMARY AND CONCLUSIONS

In this paper we discussed geometric awareness, temporal coherence, and dynamic pre-fetching techniques for improving speed and accuracy of the Voxmap PointShell collision detection method for 6-DOF haptic rendering. The desired order-of-magnitude improvement in spatial accuracy was realized, at a cost of reduced haptic fidelity that is proving acceptable. Results from performance tests conducted on large datasets were presented.

Additional VPS capabilities for distance fields and surface offsetting were discussed that can be used to enhance collision detection and response, and also provide capabilities for path planning and other proximity related applications.

The use of VPS for multiple moving objects and shared objects in a collaborative virtual environment was discussed, as were collaboration related issues unique to voxel-based haptics.

We believe that haptic feedback will always remain a powerful adjunct to visual feedback, especially in busy environments with much occlusion.

REFERENCES

[1] Adams, R. and Hannaford, B. “A Two-Port Framework for the Design of Unconditionally Stable Haptic Interfaces.” Proc. IROS, Anaheim CA, 1998.

[2] Borgefors, G., “Distance Transformations on Digital Images”, Computer Vision Graphics Image Processing, v34, pp. 344-371, 1986.

[3] Borro, D., Garcia-Alonso, A., and Matey, L., “Approximation of Optimal Voxel Size for Collision Detection in Maintainability Simulations within Massive Virtual Environments” Computer Graphics Forum, 23(1), p.13, Mar 2004.

[4] Breen, D.E., Mauch, S., and Whitaker, R.T., “3D Scan Conversion of CSG Models into Distance, Closest-Point and Colour Volumes” *Volume Graphics*, Chen, M., Kaufman, A.E., Yagel, R. (eds.), Springer, London, Ch. 8, pp. 135-158, 2000.

[5] Buttolo, P., Oboe, R., Hannaford, B., and McNeely W., “Force Feedback in Shared Virtual Simulations.” Proc MICAD, Paris, 1996.

[6] Choi, M. and Cremer, J., “Geometrically-Aware Interactive Object Manipulation”, Computer Graphics Forum, 19(1), pp. 65-76, 2000.

[7] Cohen, J., Lin, M., Manocha, D., and Ponamgi, M., “I-COLLIDE: An Interactive and Exact Collision Detection System for large-Scale Environments”, 1995 Symposium on Interactive 3D Graphics, pp. 189-196, 1995.

[8] Colgate, J.E., Grafing, P.E., Stanley, M.C., and Schenkel, G., “Implementation of Stiff Virtual Walls in Force-Reflecting Interfaces.” Proc. IEEE Virtual Reality Annual International Symposium (VRAIS), Seattle WA, pp. 202-208, Sept. 1993.

[9] Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M., and Manocha, D., “Six Degree-of-Freedom Haptic Display of Polygonal Models”, Proc. IEEE Visualization, pp. 139-146, 2000.

[10] Homan, D. “Virtual Reality Applications Development”, NASA JSC Annual Report for FY-1995 of the Automation, Robotics & Simulation Division, http://tommy.jsc.nasa.gov/ARSD/report_FY95/homan_fy955.html, including private correspondence.

[11] Johnson, D and Willemsen, P., “Accelerated Haptic Rendering of Polygonal Models through Local Decent”, Int. Symposium on Haptic Interfaces for VR and Teleop Systems, pp. 18-23, Mar 2004.

[12] Kim, J., Kim, H., Tay, B.K., Muniyandi, M., Jordan, J., Mortensen, J., Oliveira, M., Slater, M., and Srinivasan, M.A., “Transatlantic Touch: A Study of Haptic Collaboration over Long Distance.” Presence, 13(3), pp. 328-337, June 2004.

[13] Lin, M., “Efficient Collision Detection for Animation and Robotics”, Ph.D. Thesis, University of California, Berkeley, 1993.

[14] McNeely, W., Puterbaugh, K., and Troy, J., “Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling”, Proc. ACM SIGGRAPH, pp. 401-408, Aug. 1999.

[15] Mark, W., Randolph, S., Finch, M., Jan. Verth, J., and Taylor II, R., “Adding Force Feedback to Graphics Systems: Issues and Solutions”, Proc. ACM SIGGRAPH, pp. 447-452, Aug. 1996.

[16] Mirtich, B., “V-Clip: Fast and Robust Polyhedral Collision Detection”, ACM Transactions on Graphics, 17(3), pp. 177-208, 1998.

[17] Nelson, D. and Cohen, E., “Optimization-Based Virtual Surface Contact Manipulation at Force Control Rates”, Proc. IEEE Virtual Reality, pp. 37-44, 2000.

[18] Otaduy M. and Lin, M., “Sensation Preserving Simplification for Haptic Rendering”, Proc. ACM SIGGRAPH, pp. 543-553, Aug. 2003.

[19] Prior, A. and Haines, K., “The use of a Proximity Agent in a Collaborative Virtual Environment with 6 Degrees-of-Freedom Voxel-based Haptic Rendering.” Proc. World Haptics Conf., Pisa, Italy, pp. 631-632, Mar 2005.

[20] Renz, M., Preusche, C., Pötke, M., Kriegel, H.-P., and Hirzinger, G., “Stable Haptic Interaction with Virtual Environments using an Adapted Voxmap-Pointshell Algorithm” Proc. Eurohaptics, Birmingham, UK, 2001.

[21] Troy, J., “Haptic Control of a Simplified Human Model with Multibody Dynamics” Phantom Users Group Conf., Aspen, CO, pp. 43-46, Oct. 2000.

[22] Wan, M. and McNeely, W.A. “Quasi-Static Approximation for 6-DOF Haptic Rendering”. Proc. IEEE Visualization conference, Seattle, WA, pp. 257-262, Oct. 2003.

[23] Woo, M., Neider, J., Davis, T., and Shreiner, D., *OpenGL Programming Guide*, Version 1.2, 3rd, Addison Wesley, Reading, MA, 1999.