Incorporating Dynamic Real Objects into Immersive Virtual Environments

Benjamin Lok
University of North Carolina at Charlotte
bclok@cs.uncc.edu

Samir Naik
Disney Corporation
Samir.D.Naik@disney.com

Mary Whitton,
Frederick P. Brooks Jr.
University of North Carolina at Chapel Hill
[whitton, brooks]@cs.unc.edu

Abstract

We present algorithms that enable virtual objects to interact with and respond to virtual representations, *avatars*, of real objects. These techniques allow dynamic real objects, such as the user, tools, and parts, to be visually and physically incorporated into the virtual environment (VE). The system uses image-based object reconstruction and a volume query mechanism to detect collisions and to determine plausible collision responses between virtual objects and the avatars. This allows our system to provide the user natural interactions with the VE.

We have begun a collaboration with NASA Langley Research Center to apply the hybrid environment system to a satellite payload assembly verification task. In an informal case study, NASA LaRC payload designers and engineers conducted common assembly tasks on payload models. The results suggest that hybrid environments could provide significant advantages for assembly verification and layout evaluation tasks.

CR Categories: H.5.2 (Information Interfaces and Presentation): User Interfaces – Haptic I/O; I.3.7 (Computer Graphics): Three-Dimensional Graphics and Realism – Virtual reality; I.6.0 (Simulation and Modeling): General.

Keywords: interactions in virtual environments, collision detection, mixed reality

1. Introduction

Suppose one has a virtual model of a car engine and wants to use an immersive virtual environment (VE) to determine whether both a large man and a petite woman can readily replace the oil filter. This real world problem is difficult to solve efficiently with current modeling, tracking, and rendering techniques. Hybrid environments, systems that incorporate both real and virtual objects in a VE, can greatly assist in answering questions of this kind.

Conducting design evaluation and assembly verification tasks in immersive virtual environments (VEs) enables designers to evaluate and validate alternative designs more quickly and cheaply than if mock-ups are built and more thoroughly than can be done from drawings. Design review has become one of the major productive applications of VEs [Brooks 1999]. Virtual models can be used to study the following important design questions:

? Can an artifact readily be assembled?
? Can repairers readily service it?

In the assembly verification example, the ideal VE system would have the participant fully convinced he was actually performing a task [Sutherland 1965]. Parts and tools would have mass, feel real, and handle properly with appropriate visual and haptic feedback. The user would interact with virtual objects as if he were actually doing the task, and virtual objects would respond to the user's actions.

Both assembly and servicing are hands-on tasks and the principal drawback of virtual models — that there is nothing there to feel, to give manual affordances, and to constrain motions — is a serious one for these applications. Using a six degree-of-freedom wand to simulate a wrench, for example, is far from natural or realistic, perhaps too far to be useful. Imagine trying to simulate a task as basic as unscrewing an oil filter from a car engine in a VE! Interacting with purely virtual objects limits the types of feedback the system can provide.

Getting a virtual representation of a specific real object – such as the user's hand, specialized tools, or parts – into the VE requires specific development for modeling, tracking, and interaction. The required additional development effort, coupled with the difficulties of object tracking and modeling, lead designers to use few real objects in most VEs. Further, there are also restrictions on the types of real objects that can be incorporated into a VE. Highly deformable real objects, for example a bushy plant, would be difficult to include in a VE.

1.1. Incorporating Real Objects

We feel a *hybrid environment* system, one that incorporates representations of dynamic real objects into the VE, would assist in providing natural interactivity.

Dynamic real objects are defined as objects that can deform, change topology, and change appearance. Examples include a socket wrench, clothing, and the human hand. For a many types of VEs, incorporating dynamic real objects would provide improved affordance matching and tactile feedback.

Incorporating real objects is defined as being able to see, and have virtual objects react to, the virtual representations of real objects. The challenges are visualizing the real objects in the VE and managing the interactions among the real and virtual objects.

In this work, we extend the definition of an *avatar* to include a virtual representation of any real object, including the participant. These real-object avatars are registered with, and ideally have the same shape, appearance and motion as, the real object. They also provide haptic cues and physical restraints — the types of feedback that purely virtual objects can't provide.

We believe spatial cognitive tasks would benefit from incorporating real objects. These tasks require problem solving while manipulating objects and maintaining mental model of spatial relationships among them. With the capability of having real objects interact with virtual models, designers can see if there is enough space to reach a certain location (Figure 1) all while handling real parts, real tools, and the variability among participants.

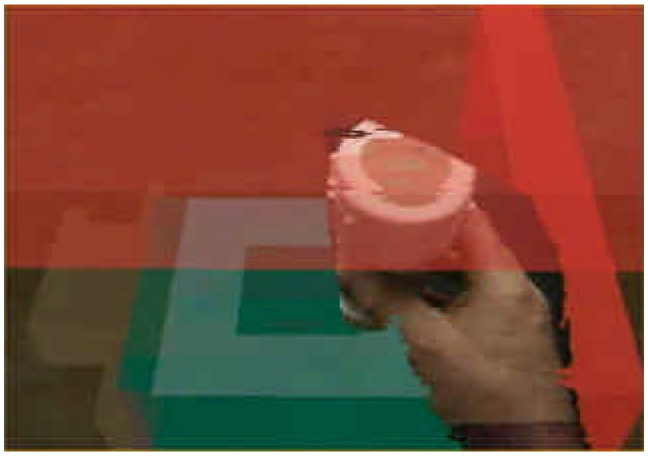


Figure 1 - A hybrid environment detects collisions between real objects (PVC pipe and user) and virtual objects (payload models)

1.2. Approach

We use a hybrid environment system that uses image-based object reconstruction algorithms to generate real-time virtual representations, avatars, of real objects. The participant sees avatars of himself and real objects visually incorporated into the VE. Further, the participant handles and feels the real objects while interacting with virtual objects.

The system models each object as the visual hull derived from multiple camera views, and then texture-maps onto the visual hull the object's image from a HMD mounted camera. The resulting virtual representations or avatars are visually combined with virtual objects with correct obscuration.

We then developed algorithms to use the virtual representations in virtual lighting and in physically-based mechanics simulations. This includes new collision-detection and collision-response algorithms that exploit graphics hardware for computing results in real time. This type of interaction allows the real-object avatars to affect simulations such as particle systems, cloth simulations, and rigid-body dynamics.

In an exploratory study, four payload design experts from NASA Langley Research Center (NASA LaRC) used the hybrid environment system to evaluate an abstracted version of a payload assembly task. The participants' experiences with the system showed anecdotally the effectiveness of handling real objects when interacting with virtual objects. We expect hybrid VEs to expand the types of tasks and applications that would benefit from immersive VEs by providing a higher-fidelity natural interaction.

2. Previous Work

Our goal is to populate a VE with virtual representations of dynamic real objects. This involves two primary components: capturing object information and having virtual systems interact with the captured object data. We review current algorithms for capturing this information, then look at methods to use the captured data as part of a virtual system.

2.1. Incorporating Real Objects into VEs

Applications that incorporate real objects must capture the shape, surface appearance, and motion of the real objects. Object material properties and articulation may also be of interest. Prebuilt models are usually not available for specific real objects. Making measurements and then using a modeling package is laborious for complex static objects, and near impossible for capturing all the degrees of freedom of complex dynamic objects.

Creating a virtual version of a real scene has many applications. For example, capturing 3-D models of archeological sites, sculptures, and objects allows new methods of education, visualization, archiving, and exploration [Levoy *et al.* 2000]. Generating novel views of sporting events from several cameras were used to enhance television coverage of Superbowl XXXV [Baba et al. 2000]. The Office of the Future project, described by Raskar et al., generates 3-D models of participants from multiple camera images [1998] for telecommunications

Real-time algorithms simplify object reconstruction by restricting the inputs, making simplifying assumptions, or accepting output limitations. This allows the desired model to be computed at interactive rates.

For example, the 3-D Tele-Immersion reconstruction algorithm by Daniilidis, *et al*., restricts the reconstructed volume size and number of input cameras so that usable results can be computed in real time using their dense-stereo algorithm [2000].

For some applications, precise models of real objects are not necessary. A simplification is to compute approximations of the objects' shapes, such as the visual hull. A shape-from-silhouette concept, the *visual hull*, for a set of objects and set of *n* cameras, is the tightest volume that can be obtained by examining only the object silhouettes, as seen by the cameras [Laurentini 1994].

At SIGGRAPH 2000, Matusik, *et al*., presented an image-based visual hull algorithm, "Image Based Visual Hulls" (IBVH), which uses image-based rendering (IBR) algorithms to calculate the visual hull at interactive rates [2000]. Further, the IBVH algorithm computes visibility, coloring, and creating polygonal models of visual hulls [2001].

Similar in spirit to this work are augmented reality systems [Feiner 1993], which visually incorporates a few virtual objects with the real world. Our work focuses on incorporating a few real objects into a virtual environment, for example, training for assembling a virtual payload in outer space with real tools and parts.

2.2. Collision Detection

Detecting and resolving collisions between moving objects is a fundamental issue in physical simulations. Our work not only detects collisions between the VE and a representation of the user, but also between the VE and any real objects the user introduces into the system.

Collision detection between virtual objects is an active area of research. Efficient and accurate packages, such as Swift++, detect collisions between polygonal objects, splines, and surfaces [Ehmann and Lin 2000]. Hoff, *et. al.*, use graphics-hardware accelerated functions to solve for collisions and penetration information [2001]. Other approaches to collision detection between real and virtual objects first create geometric models of the rigid-body real objects and then use standard collision detection approaches [Breen *et. al.* 1996].

Ideally, a participant would interact with objects in the VE in the same way he would in a real world situation, i.e. using his hands, body, and tools to manipulate objects in the environment. As a step toward this goal, some VE systems provide tracked, instrumented real objects as input devices. Common devices include articulated gloves with gesture recognition or buttons (Immersion's Cyberglove), mice (Ascension Technology's 6D Mouse), or joysticks (Fakespace's NeoWand).

Another approach is to engineer a device for a specific type of interaction to improve interaction affordance. For example, tracking a toy spider registered with a virtual spider [Hoffman *et. al.* 1997] or tracking a doll's head and props (cutting plane and stylus) to enable doctors to more naturally visualize MRI data [Hinckley 1994] enhances the VE interaction. However, this specialized engineering can be time-consuming and the results are often application specific.

[Hand 1997] and [Bowman and Hodges 1997] provide good summaries of studies of interaction in VEs.

3. Real Object Reconstruction

We use a real-object reconstruction algorithm that exploits the tremendous recent advances in graphics hardware performance to render the visual hulls of real objects [Lok 2001]. Here, we provide a volume query overview as background for Section 4.

3.1. Capturing Real Object Shape

The reconstruction algorithm, takes multiple, live, fixed-position video camera images, identifies newly introduced real objects in the scene (*image segmentation*) and then computes a novel view, corresponding to the user's view direction, of the real objects' shape (performing volume query).

When a new frame is captured, it is compared to a reference image of an "empty" scene (only the background) to create a difference image. This background subtraction with thresholding identifies the pixels (labeled *object pixels*) that correspond to newly introduced objects. This is done for each camera at every frame and produces a set of object pixels ($S(O_i)$). The visual hull is the intersection of the projected right cones of the 2D object pixel maps. The visual hull is a conservative approach that always encompasses the real objects.

3.2. Volume Query

To volume query a point asks, *given a 3D point (P), is it within the visual hull (VH) of a real object in the scene?* P is within the visual hull if and only if P projects onto an object pixel in each camera (defined by perspective matrix C_i)

$$P \text{ ? } VH_{\text{real objects}} \text{ iff ? } i,\ P = C_i^{-1} O_i \qquad (1)$$

To render the visual hull from a novel viewpoint, the volume of the new view frustum is queried against the visual hull. We sample the view frustum with a series of planes ordered front to back and orthogonal to the view direction. By rendering a large primitive such as a plane, we volume query for many 3D points in a massively parallel manner.

To accelerate the volume query process, we make use of the texture mapping and stencil buffer hardware available in most graphics accelerator cards. For each plane, the *i*th's camera's texture is projected onto the plane and a pixel's stencil buffer is incremented if an object pixel projects onto the plane at that pixel location ($P = C_i^{-1}O_i$). After all *n* textures have been projected onto the plane, only the pixels with a stencil value equal to *n* are kept (? i, $P = C_i^{-1} O_i$). They represent the points on the plane that are within a visual hull. The volume query technique results in a correctly *z*-buffered first-visible surface representation of the visual hull from the novel viewpoint. Traditional rendering of the VE will correctly incorporate the result.

While planes are used for novel viewpoint reconstruction, it is possible to volume query with any primitive. We use this in detecting intersections between real and virtual objects.

4. Visual Hull – Virtual Object Collision Detection and Response

The *collision detection* and *collision response* algorithms enable real objects to be dynamic inputs to simulations running in the application. Combined with lighting and shadow rendering algorithms, they provide a more natural interface with the VE. That is, participants can use real objects to interact with virtual objects as if the environment were real.

For example, Figure 5 shows a participant parting a virtual curtain to look out a virtual window. The interaction between the real hand and virtual cloth involves first detecting the collision between hand and cloth, and then the cloth simulation's appropriately responding to the collision. Collision detection occurs first and computes information used by the application to compute the appropriate response.

The laws of physics resolve collisions between real objects; standard collision detection packages handle collisions between virtual objects. We present an image-space algorithm to detect and allow virtual objects to plausibly respond to collisions with real

4.1. Collision Detection

Standard collision detection algorithms detect collisions among objects defined as geometric models. Since our reconstruction algorithm does not generate a geometric model of the visual hull, we needed new algorithms to detect collisions between the real-object avatars and virtual objects. Similar to how the object reconstruction algorithm volume-queries the novel view frustum, the collision detection algorithm tests for collisions by performing a volume query with the primitives composing the virtual objects.

The real-virtual collision detection algorithm takes as inputs a set of *n* live camera images and a set of triangles defining the virtual objects. It outputs:

? (CP_i) – set of points on the virtual object surface that are within a real-object avatar.

It also estimates, within some tolerance, the following:

? (CP_{obj}) – point of first contact on the virtual object
? (CP_{hull}) – point of first contact on the visual hull
? (V_{rec}) – recovery vector and
? (D_{rec}) – distance to translate the virtual object out of collision with the real-object avatar
? (N_{hull}) – surface normal at the point of visual hull contact.

4.1.1. Assumptions

A set of simplifying assumptions makes interactive-time real-virtual collision detection a tractable problem.

Only virtual objects can move or deform as a consequence of collision. Because the real-object avatars are registered with the real objects, and virtual objects cannot physically affect the real objects themselves, the system does not handle virtual objects affecting real objects due to collision.

Both real objects and virtual objects are considered stationary at the time of collision. Real-object avatars are computed anew each frame. No information, such as a centroid of the visual hull, is computed and retained between frames. Consequently, no information about the motion of the real objects, or of their hulls, is available to the real-virtual collision detection algorithm. This means the algorithm cannot determine *how* or *when* the real and virtual objects came into collision. It simply suggests a way to move the virtual object out of collision.

There is at most one collision between a virtual object and the real object visual hull at a time. If the real object and virtual object intersect at disjoint locations, we apply a heuristic to estimate the point of first contact. This is due to the inability to backtrack the real object to calculate the true point of first contact.

The real objects that contribute to the visual hull are treated as a single object. Although the real-object avatar may appear visually as multiple disjoint volumes, e.g., two hands, computationally there is only a single visual hull representing all real objects in the scene.

Collisions are detected relatively shortly after a virtual object enters the visual hull, and not as the virtual object is exiting the visual hull. This assumes the simulation time step (frame rate) is fast compared to the dynamics of the objects, and thus moving the virtual object along the vector compute by our algorithm will approximate backing the virtual object out of collision.

4.1.2. Approach

There are two steps to managing the interaction of virtual objects with real-objects avatars. The first and most fundamental operation is determining whether a virtual object, defined by a set of geometric primitives representing its surface, is in collision with a real object, represented by its visual hull.

If a virtual object is found to be in collision with a real object, the next step is to reduce or eliminate any unnatural penetration. Whereas the simulation typically has additional information on the virtual object, such as velocity, acceleration, and material properties, we do not have this information for the real object. Recall that we do not track, or have models of, the real object. To the reconstruction system, the real object is only an occupied volume.

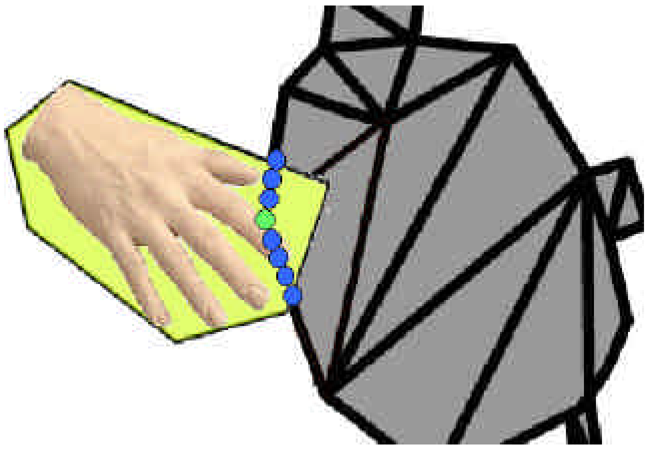


Figure 2 – The first step is to detect if any points of the virtual objects (teapot) are within the visual hull of the real objects (hand) – the set of collision points, CP_i (dots).

Since it is not possible to backtrack the real object, it is not possible to determine the exact time of collision and the points of first collision for the virtual object or the real object. If a collision occurred, we seek only to *recover* from any erroneous interpenetration by moving the virtual object out of collision.

4.1.3. Detecting Collisions

The real-virtual object collision detection algorithm performs a volume query to determine which points on the surface of virtual objects, if any, are within a visual hull. These collision points make up the set CP_i. The set of collision points is a sampling of the virtual object surface.

In novel viewpoint reconstruction (Section 3), the points in the view frustum volume were volume-queried to determine which were inside the visual hull. Collision detection volume queries with the triangles defining the virtual object's surface. If any part of a triangle lies within the visual hull, the virtual object is intersecting a real-object avatar, and a collision has occurred (Figure 2).

As in the novel viewpoint reconstruction, the algorithm first sets up n projected textures – one each for the n cameras and using that camera's image, object-pixel map, and projection matrix. Performing a volume query with each triangle involves rendering the triangle n times, once with each of the projected textures. During each rendering pass a pixel's stencil buffer is incremented if the pixel is 1) part of the triangle being scan converted *and* 2) textured by a camera's object pixel. After the n rendering passes, the stencil buffer will have values in the range of $[0...n]$. A collision occurs if any pixel has a stencil buffer value $= n$, indicating some part of a triangle, and in turn a virtual object, is within a visual hull.

If the triangle is projected 'on edge' during scan-conversion, the sampling of the triangle surface during a volume query will be sparse, and collision points could be missed. No one viewpoint will be optimal for all triangles. To address this, the algorithm performs a volume query with each triangle in its own viewport, scaled so that the triangle's projection maximally fills the viewport. To do this, each triangle is rendered from a viewpoint along the triangle's normal, and the view direction that is the inverse of the triangle's normal (Figure 3).

The sizes of the viewport and triangle affect the scan conversion sampling for performing a volume query, and thus collision detection accuracy.

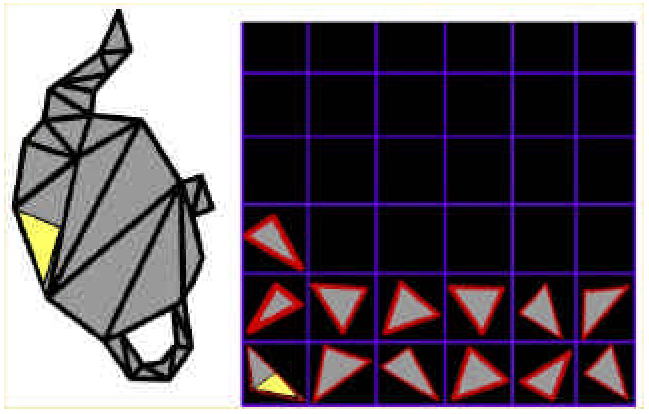


Figure 3 – The algorithm performed a volume query with each primitive in its own portion of the rendering window.

After a volume query is performed on all the triangles, the frame buffer is read back and pixels with a stencil value of *n* represent points of collision between the visual hull and the triangle. These 'collision' pixels are unprojected from screen space coordinates (u, v, depth) to world space coordinates (x, y, z). These 3-D points form a set of collision points, CP_i, for that virtual object.

The real-virtual collision detection algorithm returns whether a collision exists and a set of collision points for each triangle. How a simulation utilizes this information is application dependent. This division of labor is similar to other collision detection algorithms. We provide a suite of tools to assist in moving the virtual object out of collision with the real object.

4.2. Recovery from Interpenetration

We present one approach to use the collision information to generate a plausible response for the virtual object. As stated before, the simplifying assumptions that make this problem tractable also make the response data approximations.

The first step is to try to move the virtual object out of collision with the visual hull. We estimate the point of first contact on the virtual object, CP_{obj}, to be the collision point farthest from the virtual object's reference point, RP_{obj}. The default RP_{obj} is the center of the virtual object. CP_{obj} is not guaranteed to be the point of first collision due to our inability to backtrack the visual hull objects. CP_{obj} is not guaranteed to be unique as there may be several collision points the same distance from RP_{obj}. If multiple points are the same distance, we arbitrarily choose one of the points from the CP_i set for subsequent computations.

A *recovery vector*, V_{rec} – defined as the vector from CP_{obj} to RP_{obj} – is our estimate of the direction with possibly the shortest distance to move the virtual object out of collision. This vector works well for most objects. The simulation can specify a V_{rec} for virtual objects a constrained motion, such as a hinged door, for better, object-specific results.

V_{rec} crosses the visual hull boundary at the *hull collision point*, CP_{hull}. CP_{hull} is an estimate of the point of first contact on the visual hull, and to where CP_{obj} will be backed out.

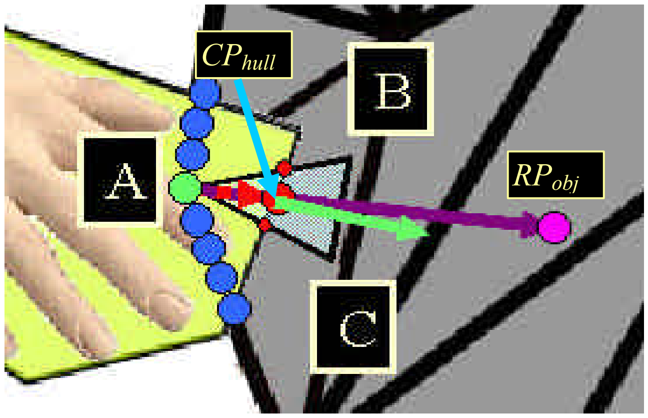


Figure 4 – CP_{obj} (A) is the farthest collision point from the virtual object's reference point (RP_{obj}). This is used to find the visual hull collision point, CP_{hull}, recovery vector (V_{rec} = A->CP_{hull}), and recovery distance (D_{rec}=|V_{rec}|).

To find CP_{hull}, V_{rec} is searched from RP_{obj} towards CP_{obj} for the first point within the visual hull. This is done by performing a volume query on an isosceles triangle *ABC*, $A = CP_{obj}$ and the base, *BC*, is bisected by V_{rec}. Angle *BAC* (at CP_{obj}) is set small (10º) so that *AB* and *AC* intersect the visual hull near CP_{hull}, and the height is set relatively large (5 cm) so the triangle base is likely to be outside the visual hull. The triangle dimensions can be altered to fit the primitive. A volume query is performed on *ABC* in the entire window's viewport and from a viewpoint along the triangle normal such that V_{rec} lies along a scan line. CP_{hull} is found by stepping along the V_{rec} scan line, starting at the base of *ABC*, to the first pixel within the visual hull (stencil buffer = *n*). Unprojecting the pixel from screen to world space yields CP_{hull}.

The *recovery distance*, D_{rec}, is the distance between CP_{obj} and CP_{hull}, and is the distance along V_{rec} required to move CP_{obj} outside the visual hull. It is not necessarily the *minimum separation distance*, as is found in other collision detection packages [Ehmann and Lin 2000], nor does it guarantee removing the virtual object completely from collisions with a visual hull.

If the visual hull collision point surface normal, N_{hull}, is required by the application for collision response, we locate four points on the visual hull surface near CP_{hull}. Stepping along *BA* and *CA* (Figure 4) and finding where they intersect the visual hull boundary determines points *I* and *J*. A second triangle, *DAE*, is constructed that is similar to *ABC* and lies in a plane roughly perpendicular to *ABC*. Points *K* and *L* are located by stepping along *DA* and *EA*. N_{hull} is the cross product of *IJ* and *KL*.

Figure 6 are frames taken from a dynamic sequence in which a virtual ball is bouncing around between a set of real blocks. N_{hull} is used in the computation of the ball's direction after collision.

4.3. Implementation

4.3.1. Hardware

The algorithms were implemented in a system that reconstructs objects within a 1.6 m x 1.3 m x 1 m volume above a tabletop. [Lok 2001] described the setup more extensively. The system used four NTSC cameras (720 x 243 resolution – 1 field of NTSC), three wall-mounted and one mounted on the user's Virtual Research V8 HMD (640 x 480 display resolution).

The participant was tracked with the UNC HiBall, a scalable wide-area optical tracker. The cameras were connected to an SGI Reality Monster system. A completely PC based solution is possible as they can now easily handle the computation and bandwidth requirements.

We used five SGI graphics pipes: a *parent pipe* to render the VE and assemble the reconstruction results, a *video pipe* to capture video, two *reconstruction pipes* for performing a volume query, and a *simulation pipe* to run simulation and collision detection. The reconstruction was done at 320x240 to reduce the pixel fill rate, and the results scaled to 640x480.

The reconstruction system runs at 15-18 FPS for a volume 0.7 m (1.5 cm spacing between planes) in front of the user.

The collision detection and response routines run on a dedicated *simulation pipe* on the SGI. At each real-virtual collision time-step, the simulation pipe performs the image segmentation operation to obtain new object-pixel maps.

For optimization, collision detection is first done between the visual hull and the virtual objects' axis-aligned bounding boxes. For the virtual objects whose bounding box was in collision with the visual hull, a per-triangle volume query is done. If collisions exist, the simulation is notified and passed the set of collision points. The simulation managing the behavior of the virtual objects decides how it will respond to the collision. Then the response algorithm computes the recovery vector, distance, and surface normal.

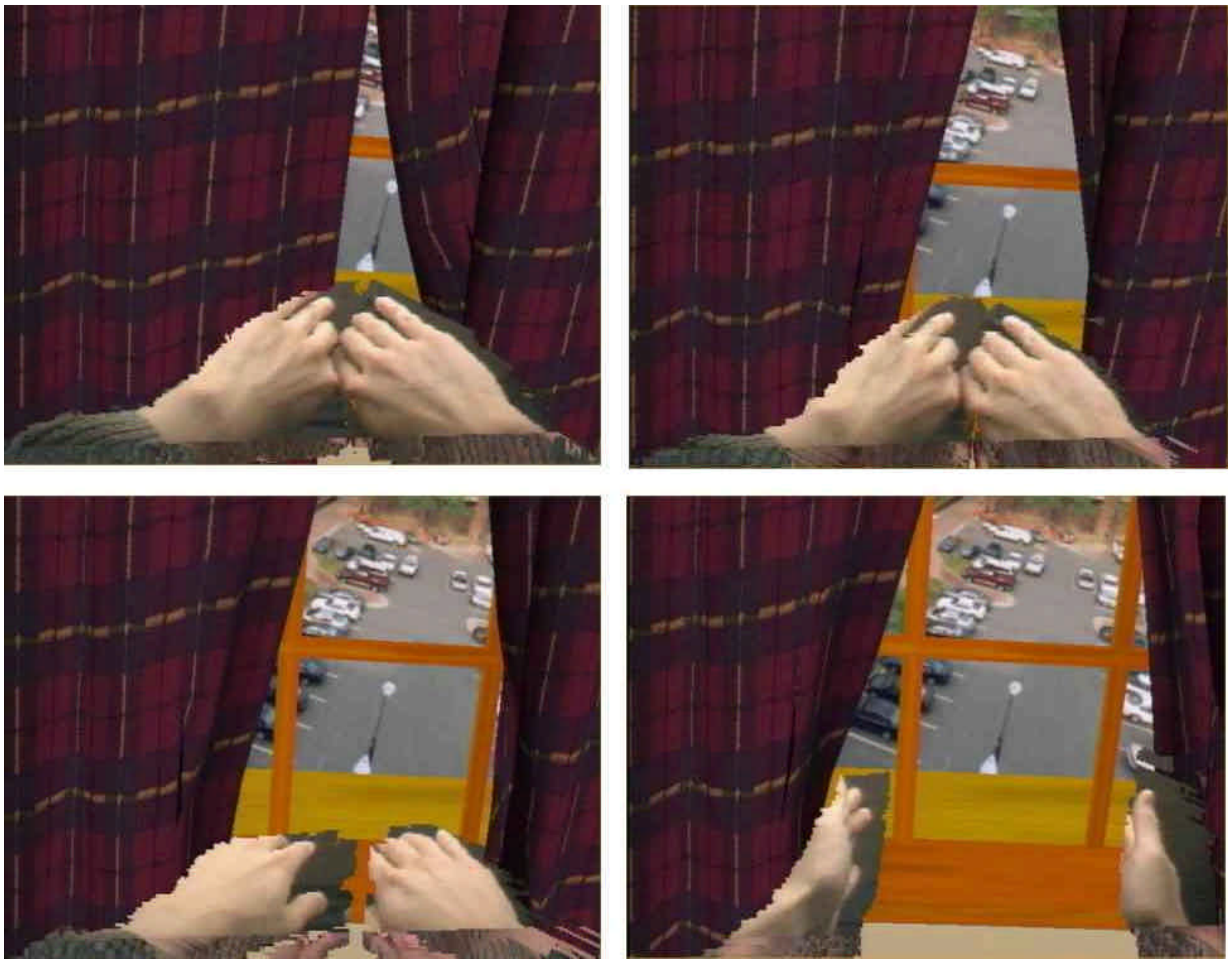


Figure 5 – Sequence of images with the user interacting with virtual curtains to look out the window.

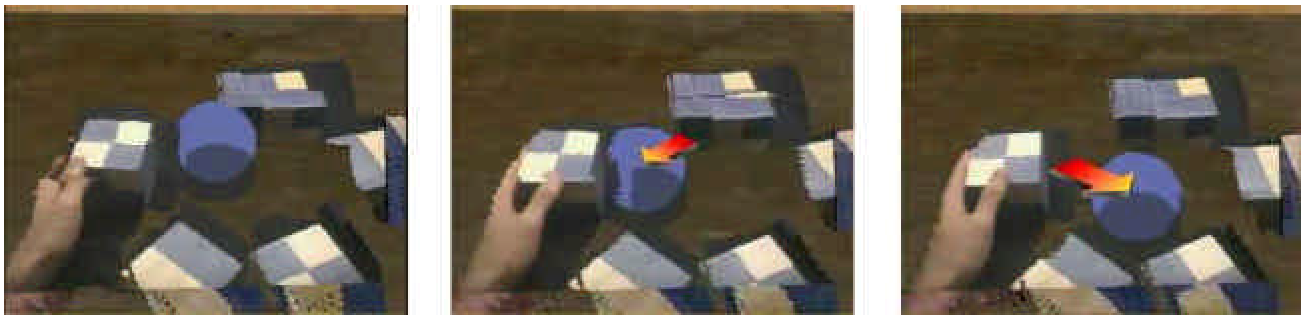


Figure 6 – A virtual ball bouncing off of real objects. The overlaid arrows show the balls motion between images.

4.3.2. *Performance*

For collision detection, given *n* cameras, virtual objects with *m* triangles, and *u* x *v* viewports in an *x* x *y* window: the geometry transformation is (*n* x *m*) triangles, pixel fill is (*n* x *m* x *u* x *v*)/2 pixels, and (*x* x *y*) pixel readbacks and compares per frame. For collision response, the transformation cost is two triangles per virtual object in collision and a pixel fill of (*x* x *y* x *n*) per frame.

The curtains in the hybrid environment in Figure 5 were composed of 720 triangles. The volume query was performed in 10 x 10 pixel viewports in a 400 x 400 window for collision detection, which ran at 6 frames per second. The collision detection work was 13,000 triangles transformations per second, 648,000 pixels, and 160,000 pixel readbacks and compares per second. The collision response work was 2 triangles and 480,000 pixels per virtual object in collision. As this was a first implementation, there are many optimizations that should improve performance.

4.3.3. *Accuracy*

The collision detection accuracy is affected by image segmentation, camera setup and models, and the use of visual hulls. The errors introduced in these stages are covered in [Niem 1997] and [Lok 2002]. Here, we examine only the effect of the volume query viewport and primitive size on collision detection accuracy.

The spatial sampling frequency is proportional to the size of the volume query viewport. The accuracy of collision detection for a *u* x *v* resolution viewport (if *u* = *v*, viewport layout is easier) and a triangle with *c* x *d* bounding box (in world space) is *c*/*u* by *d*/*u*. The accuracy is the two longest dimensions of the primitive divided by the viewport horizontal size (again assuming *u* = *v*).

One can volume query with primitives at a higher resolution (larger viewports), producing a higher number of more closely spaced collision points with less error. A larger viewport also means that fewer volume queries with triangles can be performed in the collision detection window. If all the primitives can not be allocated their own viewport in a single frame buffer, then multiple volume query and read-back cycles will needed to test all primitives.

Hence, there is a speed-accuracy tradeoff in establishing the appropriate level of parallelism: the more viewports, the faster the algorithm executes, but the lower the pixel resolution available for the collision detection (which may result in missed collisions).

The size of virtual object triangles will vary, but typical tabletop objects had triangles less than 2 cm, which would have 0.2 cm x 0.2 cm collision point detection error in 10 x 10 pixel viewports. The curtain system had a collision detection resolution of 0.75 cm x 0.3 cm. These values are the spatial frequency for performing a volume query and provide the maximum error to finding a collision point.

For collision response, the accuracy of the CP_{hull} point impacts D_{rec} and N_{hull}. The error in finding CP_{hull} along the V_{rec} is the length of triangle *ABC*'s major axis divided by the horizontal length of collision response window (assuming a square window). With a 400 x 400 collision detection window, this results in .0125 cm error for detecting CP_{hull}. The accuracy of N_{hull}, depends on the surface topology (affected by camera resolution), the distance from points *I,J,K,* and *L* to CP_{hull}, and the distance from CP_{hull} to CP_{obj}. We estimate that collision response results have at most 0.75 cm error. This is comparable to object reconstruction error, which is estimated at 0.5 cm for visual hull shape.

4.4. Algorithm Extensions

Figure 5 is a sequence of frames of a user pushing aside a virtual curtain with his hands. This shows using the algorithm with a deformable virtual object with constrained motions (a specified V_{rec} in the collision response). When trying to move individual cloth nodes out of collision, the motion is constrained in the vector direction (V_{rec}).

The algorithm can perform a volume query with primitives other than surface boundaries. We hypothesize that using primitives which represent distance fields could aid in visualizing thermal radiation of real objects onto virtual objects, magnetic fields of real objects, or barriers in a motion planning simulation. We have prototyped incorporating real-object avatars into other physical simulations, including lighting, shadowing, and particle systems.

5. NASA Case Study

We worked with space shuttle payload designers at NASA Langley Research Center (NASA LaRC) to investigate how using hybrid environments could assist them in evaluating payload designs and assembly layouts. Information reported in this paper concerning the designers' motivations, comments, and suggestions are taken directly from oral or written responses to surveys, interviews, and informal remarks made during experiments and discussions.

Given virtual models of complex multipart devices such as satellites, designers want to determine if assembling the device is physically possible. Answering this question involves managing parts, tools, and people with a large variance in shape. Space planning errors can have a significant impact in terms of money, scheduling, and personnel.

NASA LaRC designers receive payload subsection CAD models from their subcontractors early in the design stage, before anything gets built. They would like to use these models to investigate assembly, layout, and integration. Changes in the early project stages are substantially cheaper in money, time, and personnel than fixes in later stages.

Since different subsystems are separately subcontracted out, the integration stage always generates compatibility and layout issues. Even with the greatest care in the specification of subsystem designs, it is difficult to perceive the integration of the subpayloads, as the complexity and nuances of each component are understood well by only a particular group. For example, attaching external cables is a common final integration task, and the designers described several occasions when they encountered spacing problems during the final cable attachment step. The payloads had conformed to specifications, but the reality of attaching the cables showed inadequate space for hands, tools, or parts. Layout issues result in schedule delays, equipment redesign, or makeshift engineering fixes.

Later in the development cycle, simplified physical mock-ups are manufactured for design verification and layout. The assembly procedure is documented in a step-by-step instruction list. The designers recounted several occasions when the limited fidelity of mock-ups and assembly documents caused significant problems to slip through to later stages. They also suggested that interacting with virtual models early in the design cycle would enable training additional technicians on critical assembly stages.

A hybrid VE system would enable designers to test configurations using the final assembly personnel, real tools and parts. We hypothesize that such a hybrid VE would be a more effective system for evaluating hardware designs and planning assembly than a purely virtual one.

5.1. Payload Spacing Study

During a visit to the NASA LaRC facilities, we were shown a weather imaging satellite (CALIPSO) and a light imager unit on the satellite called the photon multiplier tube (PMT) (Figure 7). We used CAD models of the PMT for our case study, and abstracted a task that was similar to common assembly steps.

5.1.1. Assembly Task Description

The PMT model and two other fictional payloads (payload A and payload B) were rendered in the VE. The system performed object reconstruction (on the user, tools, and some parts) and collision detection among the virtual payloads and the real-object avatars. Virtual objects in collision were rendered red (Figure 1).

The task, diagramed in Figure 9, was to screw a cylindrical shield (mocked-up as a PVC pipe - Figure 8) into a receptacle (Figure 10) and then plug a power connector into an outlet inside the shield (Figure 11). If the participant asked for additional assistance, we provided tools to aid in the task (Figure 12).

5.1.2. Study Procedure

Four NASA LaRC payload designers participated in the case study. First, we provided task information in approximately the same manner as in actual design evaluation. The participants were asked how much space was needed – and how much would actually be allocated – between the PMT and payload A.

Each participant then performed the pipe insertion and power cable attachment procedure in the hybrid system. After a period of VE adjustment, participants picked up the pipe and eased it into the center cylindrical assembly while trying to avoid colliding with any of the virtual payloads. After the pipe was lowered into the cylindrical shaft of the PMT, they snaked the power cord down the tube and inserted it into the outlet.

If the participant asked for more or less space between the PMT and payload A, the experimenter could dynamically adjust the spacing. This allowed quick evaluation of different spatial configurations of the two payload subassemblies.

5.2. Results

For the study, the pipe had a length of 14 cm and a diameter of 4 cm. Table 1 summarizes the data collected.

Between Payload A and PMT	#1	#2	#3	#4
(Pre) How much space is necessary?	14.0	14.2	15.0-16.0	15.0
(Pre) How much space would you allocate?	21.0	16.0	20.0	15.0
Actual space required in VE	15.0	22.5	22.3	23.0
(Post) How much space would you allocate?	18.0	16.0 adjust tool	25.0	23.0

Table 1 – LaRC participant responses and task results (cm)

Space is scarce and the engineers were stingy with it. Participant #1 was able to complete the task without using any tool, as the power cable was stiff enough to force into the outlet. Since an aim was to impress upon the participants the unforeseen possibility of requiring tools in assembly or repair, we used a more flexible cable for the remaining participants.

While trying to insert the power cable, participants #2, 3, and 4 noted they could not complete the task. The more flexible power cable could not be snaked down the pipe and inserted into the outlet without some device to help push the connector when it was inside the pipe. This was because the pipe was too narrow for the participant's hands. When asked what they required, they all remarked they wanted a tool to assist in plugging in the cable. They were handed a tool (set of tongs). They were then able to complete the power cable insertion task. This required increasing the spacing between the PMT and Payload A from 14 cm to an average of 24 cm to avoid collisions.

Whereas in retrospect it was obvious that the task would not be easily completed without a tool, none of the designers anticipated this requirement. We believe the manner the assembly information was provided (diagrams, assembly documents and drawings), made it difficult for designers – even though each had substantial payload development experience – to identify subtle assembly integration issues. On average, the participants under allocated an average of 5.6 cm between the payloads.

Accommodating tools extemporaneously in a VE session, without additional modeling or development, enabled easy evaluation of multiple layouts, approaches, and tools. The presence of the pipe threads and cable socket provided important motion constraints that aided in interacting with these objects.

Physical mock-ups are costly to build, require substantial time to create, and have varying degrees of fidelity to the final payload. These characteristics reduce their use early in the design evaluation stage. In the early design phases, hybrid VEs can provide a cheaper and quicker alternative for evaluating designs and layouts than mock-ups.

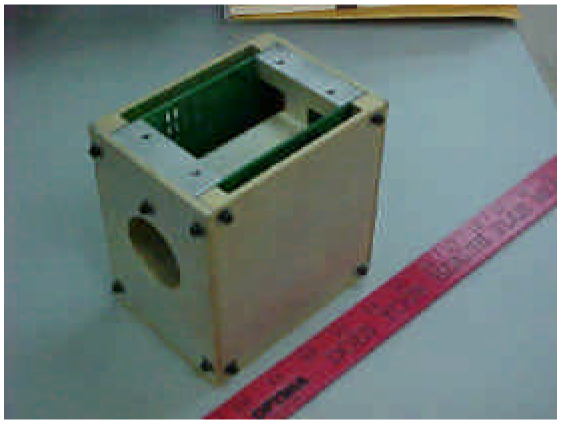
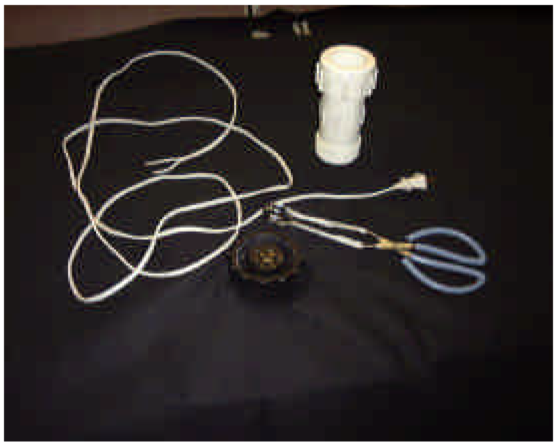
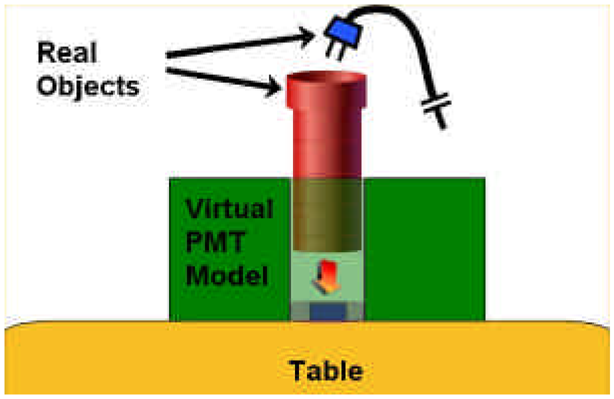
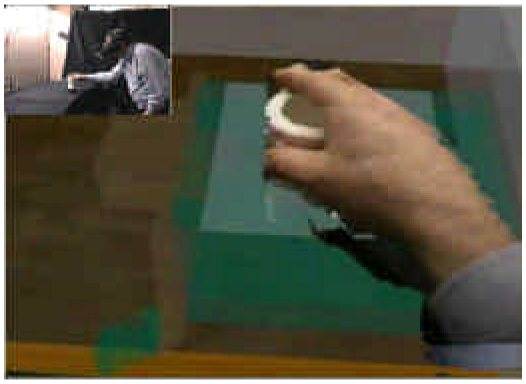
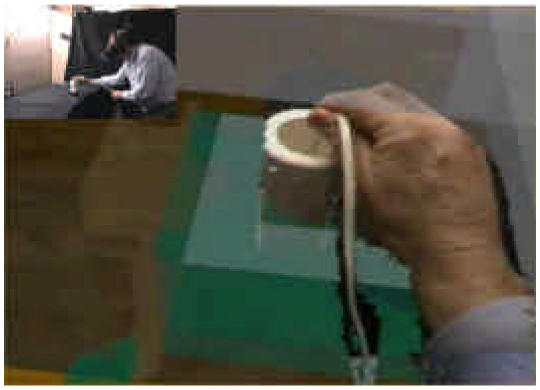
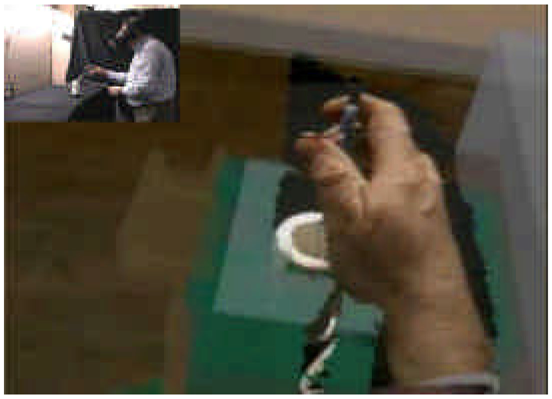
5.3. Debriefing

The participants were *extremely* surprised that both a tool and substantial additional space were required. When they discovered the required spacing was much more than the amount they allocated, they immediately focused on the potential time and schedule savings of finding such an error at the design stage. The participants commented that the financial cost of the spacing error could range from moderate (keeping personnel waiting until a design fix was implemented) to extreme (launch delays).

The virtual model was not very detailed, and the visual contrast between real and virtual objects was rather obvious. Yet, participants were observed to make concerted efforts to avoid touching the virtual model. Upon being told about his effort to avoid touching the virtual PMT box, one participant said, "That was flight hardware… you don't touch flight hardware." The familiarity and relevance of the task made it a vivid experience for the participants.

NASA LaRC payload designers remarked that VEs and object reconstruction VEs would be useful for assembly training, hardware layout, and design evaluation. They are also interested in looking at traditional VEs and virtual models to evaluate current payload integration tasks and upcoming payload designs.

There are substantial gains to be realized by using virtual models in almost every stage of payload development. Early identification of assembly, integration, or design issues can result in considerable savings in terms of time, money, and man-hours. Many of NASA LaRC tasks involve technicians interacting with a payload with tools and parts. These tasks are well suited to be simulated within a hybrid VE.

 Figure 7 – Photon Multiplier Tube (PMT) box for the CALIPSO satellite payload. We used this payload subsystem as the basis for our case study. Courtesy of NASA LaRC's CALIPSO project.	 Figure 8 – Parts used in the shield fitting experiment. PVC pipe prop, power cord, tongs (tool), and the outlet and pipe connector that was registered with the virtual model.
 Figure 9 – Cross-section diagram of task. The pipe (red) and power cable (blue) need to be plugged into the corresponding connector down the center shaft of the virtual PMT box.	 Figure 10 - The first step was to slide the pipe between the payloads and then screw it into the fixture.
 Figure 11 – After the pipe was in place, the next step was to fish the power cable down the pipe and plug it into the outlet on the table.	 Figure 12– The insertion of the cable into the outlet was difficult without a tool. Tongs were provided to assist in the plugging in the cable.

6. Conclusions and Future Work

We have developed a system for incorporating dynamic real objects into a virtual environment. This involved developing image-based hardware-accelerated algorithms for detecting collisions and providing plausible responses between virtual representations of real objects and other virtual objects. Future work would focus on improving performance, increasing accuracy, and providing better collision response information.

The present limitation in our responding to collisions follows from the inability to backtrack the motions of real objects. Retaining previous object pixel maps images, along with tracking real objects within the camera images, could enable backtracking. By looking at the shape and motion of a tracked object across several frames, information, such as object velocity, acceleration, rotation, and center of mass, could be derived. This information could allow for more accurate collision response.

We believe that many VE applications, such as training, telepresence, phobia treatment, and assembly verification, could be assisted through interacting with real objects in a VE. Our work with NASA LaRC has shown that the system could provide a substantial advantage in hardware layout and assembly verification tasks. Future work would identify tasks that would most benefit from having the user handle dynamic real objects.

7. Acknowledgements

We thank Danette Allen, Marilee Roell, Lloyd Spangler, Kerry Gough, and Glenn Hines from NASA Langley Research Center for participating in evaluating the system. We also thank Henry Fuchs, Herman Towles, David Harrison, Kurtis Keller, John Thomas, and the UNC-Chapel Hill Effective Virtual Environments group. This work was supported by the LINK Simulation and Training Foundation, ONR Virte Program, NIH National Institute of Biomedical Imaging and Bioengineering, NSF Information Technology Research BioGeometry Project, UNC-Chapel Hill, and UNC-Charlotte.

8. Bibliography

BABA, S., SAITO, H., VEDULA, S., CHEUNG, K., AND KANADE, T. 2000. Appearance-Based Virtual-View Generation for Fly Through in a Real Dynamic Scene. *VisSym '00* (Joint Eurographics – IEEE TCVG Symposium on Visualization).

BOWMAN, D., AND HODGES, L. 1997. An Evaluation of Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments. In *Proceedings 1997 ACM Symposium on Interactive 3-D Graphics*, 35-38.

BREEN, D., WHITAKER, R., ROSE, E., AND TUCERYAN, M. 1996. Interactive Occlusion and Automatic Object Placement for Augmented Reality, *Computer Graphics Forum*, 11-22.

BROOKS, F. 1999. What's Real About Virtual Reality? In *IEEE Computer Graphics and Applications*. Vol. 19, 6, 16-27.

DANIILIDIS, K., MULLIGAN, J., MCKENDALL, R., KAMBEROVA, G., SCHMID, D., AND BAJCSY R. 2000. Real-Time 3-D Tele-immersion. In *The Confluence of Vision and Graphics*, A Leonardis *et al*., Ed., Kluwer Academic Publishers.

EHMANN, S., AND LIN, M. 2000. Accurate Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching. In *Proceedings of the International Conference on Intelligent Robots and Systems*.

FEINER, S., MACINTYRE, B., AND SELIGMANN, D. 1993. Knowledge-based Augmented Reality. *Communications of the ACM,* Vol. 36, 7, 52-62.

HAND, C. 1997. A Survey of 3-D Interaction Techniques. *Computer Graphics Forum*, Blackwell Publishers, Vol. 16, 5, 269-281.

HINCKLEY, K., PAUSCH, R., GOBLE, J. AND KASSELL, N. 1994. Passive Real-World Interface Props for Neurosurgical Visualization, In *Proceedings of the 1994 SIG-CHI Conference*, 452-458.

HOFF, K., ZAFERAKIS, A., LIN, M., AND MANOCHA, D. 2001. Fast and Simple 2-D Geometric Proximity Queries Using Graphics Hardware. In *Proceedings 2001 ACM Symposium on Interactive 3-D Graphics*, 145-148.

HOFFMAN, H., CARLIN, A. AND WEGHORST, S. 1997. Virtual Reality and Tactile Augmentation in the Treatment of Spider Phobia. *Medicine Meets Virtual Reality 5*.

LAURENTINI, A. 1994. The Visual Hull Concept for Silhouette-Based Image Understanding. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, 2, 150-162.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The Digital Michelangelo Project. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, ACM, 131-144.

LOK, B. 2001. Online Model Reconstruction for Interactive Virtual Environments. In *Proceedings 2001 ACM Symposium on Interactive 3-D Graphics*, 69-72, 248.

LOK, B. 2002. Interacting with Dynamic Real Objects in Virtual Environments. Ph.D. Dissertation, Department of Computer Science, University of North Carolina at Chapel Hill.

MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER S., AND MCMILLAN, L. 2000. Image-Based Visual Hulls. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, 369-374.

MATUSIK, W., BUEHLER, C., AND MCMILLAN, L. 2001. Polyhedral Visual Hulls for Real-Time Rendering. In *Proceedings of Eurographics Workshop on Rendering 2001*.

NIEM, N. 1997. "Error Analysis for Silhouette-Based 3D Shape Estimation from Multiple Views", In *Proceedings on International Workshop on Synthetic - Natural Hybrid Coding and Three Dimensional Imaging*, Rhodos.

RASKAR, R., WELCH, G., CUTTS, M., LAKE, A., STESIN, L., AND FUCHS, H. 1998. The Office of the Future: A Unified Approach to Image-Based Modelling and Spatially Immersive Displays. In *Computer Graphics*. ACM Press, Addison-Wesley: 179-188.

SUTHERLAND, I. 1965. The Ultimate Display. In *Proceedings of IFIP '65*, vol. 2, 506.