Incorporating Dynamic Real Objects into Immersive Virtual Environments

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

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

Abstract

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 real and virtual objects within the VE, can greatly assist in studying this question.

We present algorithms to allow virtual objects to interact with and respond to virtual representations, *avatars*, of dynamic real objects. This allows dynamic real objects, such as the user, tools, and parts, to be visually and physically incorporated into the VE. The system uses image-based object reconstruction and a volume-querying mechanism to detect collisions and to determine plausible collision responses between virtual objects and the real-time 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

1.1. Motivation

Conducting design evaluation and assembly feasibility evaluation tasks in immersive virtual environments (VEs) enables designers to evaluate and validate multiple 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 believe he was actually performing a task. Parts and tools would have mass, feel real, and handle appropriately. The user would interact with every object as if he would if he were doing the task. The virtual objects would in turn respond to the user's action appropriately. Training and simulation would be optimal [Sutherland 1965].

In current VEs, almost all objects in the environment are virtual. But 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 (DOF) wand to simulate a wrench, for example, is far from 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 such a VE!

Interacting with purely virtual objects imposes several limiting factors on VEs. First, it limits the types of feedback the system can provide to the user, such as constraints and haptics and a visually faithful virtual representation, its *avatar*. Second, it hampers real objects (including the user), naturally interacting with virtual objects.

We extend our definition of an avatar to include a *virtual representation* of any real object. These real-object avatars are registered with the real object and ideally have the same shape, appearance and motion as the real object.

Getting shape, motion, and actions from real objects, such as the user's hand, specialized tools, or parts, 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. For example, highly deformable objects, such as a bushy plant, would be especially difficult to model and track.

Working with virtual objects could hinder training and performance in tasks that require haptic feedback and natural affordances. For example, training with complex tools would understandably be more effective with using real tools as opposed to virtual approximations.

1.2. Incorporating Real Objects

We believe a system that could handle *dynamic real objects* would assist in interactivity and provide visually faithful virtual representations. We define dynamic objects as real objects that can deform, change topology, and change appearance. Examples include a socket wrench set, clothing, and the human hand. For assembly verification tasks, the user, tools, and parts are typically dynamic in shape, motion, and appearance. For a substantial class of VEs, incorporating dynamic real objects would be provides improved affordance matching and tactile feedback.

We define *incorporating real objects* as being able to see and have virtual objects react to the virtual representations of real objects. The challenges are visualizing the real objects within the VE and managing the interactions between the real and the virtual objects.

We feel that spatial cognitive manual tasks would benefit with increased task performance and presence from incorporating real objects. These tasks require problem solving through manipulating and orientating objects while maintaining mental relationships among them. These are common skills required in simulation and training VEs. By having the real objects interacting with a virtual model, designers can see if there is enough space to reach a certain location or train people in assembling a model at different stages, all while using real parts, real tools, and the variability among participants.

Today, neither standard tracking technologies nor modeling techniques are up to doing this in real time at interactive rates.

1.3. 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 both himself and any real objects introduced into the scene 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 lit appearance of the real object. 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.

We wanted to apply this system to a real world problem. So we began collaborating with a payload-engineering group at NASA Langley Research Center (NASA LaRC) in Hampton, Virginia. In a first exploratory study, four experts in payload design and engineering used the reconstruction 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 that can only be achieved by incorporating real objects.

2. Previous Work

2.1. Incorporating Real Objects into VEs

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 data. We review current algorithms to capturing this information, then look at methods to use the captured data as part of a virtual system.

Applications that incorporate real objects seek to 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, 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 the 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. One 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].

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 the user, but also between the VE and any real objects the user introduces into the system.

Collision detection between virtual objects is an area of *vast* previous and current research. Highly efficient and accurate packages, such as Swift++, detect collisions between polygonal objects, splines, and surfaces [Ehmann and Lin 2000]. Hoff, *et. al.*, uses graphics-hardware accelerated functions to solve for collisions and generate penetration information [2001]. Other work on collision detection between real and virtual objects first created geometric models of the rigid-body real objects and then used standard collision detection approaches [Breen et. al. 1996].

Ideally, a participant would interact with the VE in the same way as he would in a real world situation. The VE system would understand and react to expressions, gestures, and motion. The difficulty is in capturing this information for rendering and simulation input.

The fundamental interaction problem is that most things are not real in a virtual environment. In effort to address this, some VEs provide tracked, instrumented real objects as input devices. Common interaction devices include an articulated glove with gesture recognition or buttons (Immersion's Cyberglove), tracked mouse (Ascension Technology's 6D Mouse), or tracked joystick (Fakespace's NeoWand).

Another approach is to engineer a device for a specific type of interaction, such as tracking a toy spider registered with a virtual spider [Hoffman et. al. 1997]. This typically improves interaction affordance, so that the participant interacts with the system in a more natural manner. For example, augmenting a doll's head with sliding rods and trackers enables doctors to more naturally select cutting planes for visualizing MRI data [Hinckley 1994]. However, this specialized engineering is time-consuming and often usable for only a particular type of task.

VE interaction studies have been done on interaction ontologies [Bowman and Hodges 1997], interaction methodologies [Hand 1997], and 3-D GUI widgets and physical interaction [Lindeman et. al. 1999].

3. Real Object Reconstruction

We use a real-object reconstruction algorithm that exploits the tremendous recent advances in graphics hardware to render the visual hulls of real objects [Lok 2001]. The algorithm also provides an efficient approach to detecting collisions between real and virtual objects.

In this section we provide a general overview of the algorithm to provide the background for volume querying to detect and respond to intersections between real and virtual objects.

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 of the real objects' shape (volume-querying).

When a live frame is captured, it is compared against a reference image of an "empty" scene (only the background). The difference image 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 Querying

Volume querying 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 iff P projects onto an object pixel from each camera (defined by perspective matrix C_i)

$$P \ni R_3,\ P \ni VH_{\text{real objects}} \text{ iff } \forall\ i,\ P = C_i^{-1} O_i \qquad (1)$$

To render the visual hull from a novel viewpoint, the volume of the new view frustum is queried against the visual hull. To accelerate the volume querying process, we make use of the texture mapping and stencil buffer hardware available in most graphics cards. By rendering a large primitive such as a plane, we query the volume for many 3D points in a massively parallel manner. We sample the view frustum with a series of planes orthogonal to the view direction.

For each plane, the *i*th's camera's texture is projected onto the primitive and a pixel's stencil buffer is incremented if an object pixel projects onto the primitive at that pixel's location ($P = C_i^{-1}O_i$). After each texture is projected onto the primitive, only the pixels with a stencil value equal to *n* are kept ($\forall\ i,\ P = C_i^{-1} O_i$). They represent the points in the primitive that are within a visual hull. While planes are used for novel viewpoint reconstruction, the volume querying method works for any primitive. We make use of this property in detecting intersections between real and virtual objects.

The volume querying technique results in a correctly *z*-buffered first-visible surface of the visual hull from the novel viewpoint. Rendering of the virtual environment will correctly incorporate the result.

4. Visual Hull – Virtual Object Collision Detection and Response

4.1. Overview

The *collision detection* and *collision response* algorithms, along with the lighting and shadowing rendering algorithms, enable the real objects to be dynamic inputs to simulations and provide a natural interface with the VE. That is, participants would interact with virtual objects the same way as if the environment were real.

For example, Figure 4 shows a participant parting a virtual curtain to look out a virtual window. The interaction between the real hand and virtual cloth involves first upon detecting the collision between hand and cloth, and then upon 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.

We define *interactions,* as one object affecting another. 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 the virtual objects to plausibly respond to collisions with real objects. 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.

4.2. Collision Detection

4.2.1. Overview

Standard collision detection algorithms detect collisions among objects defined as geometric models. Since the 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 volume-querying with the virtual objects primitives.

The *real-virtual* collision detection algorithm takes as inputs a set of *n* live camera images and virtual objects defined by triangles. It outputs a set of points (CP_i) on the virtual object surface that are within a real-object avatar. It also estimates, within some tolerance, the following: the point of first contact on the virtual object (CP_{obj}) and the visual hull (CP_{hull}), the recovery vector (V_{rec}) and distance (D_{rec}) to translate the virtual object out of collision with the real-object avatar, and surface normal at the point of visual hull contact (N_{hull}).

4.2.2. 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. This follows from our restrictions on virtual objects affecting real objects due to collisions.

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 defined in our algorithm will approximate backing the virtual object out of collision.

4.2.3. 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, computationally represented by its visual hull volume.

For a virtual object and real object in collision, 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, so we do not use any such information in our algorithm. Recall that we do not track, or have models of, the real object. To the reconstruction system, the real object is an occupied volume.

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.

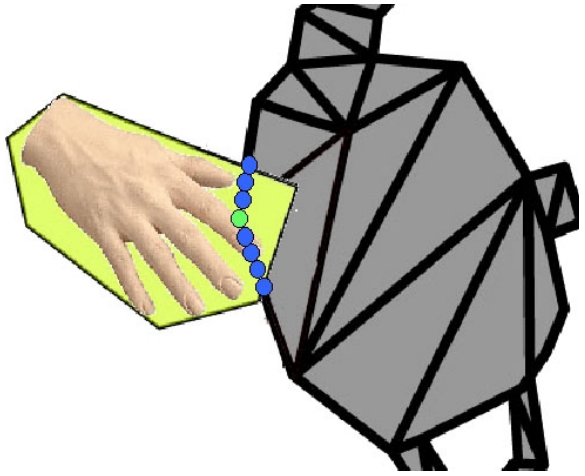
Figure 1 – The first step is to detect if any points of the virtual objects (teapot) are within the visual hull of the real objects (hand) – the set of collision points (CPi) the blue dots.

4.2.4. Detecting Collisions

Collision points, CP_i, are points on the surface of the virtual object that are within the visual hull. As the virtual surfaces are continuous, the set of collision points is a sampling of the virtual object surface.

The real-virtual object collision detection algorithm uses volume-querying. In novel viewpoint reconstruction, 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 to determine if any parts of the surface is inside the visual hull. If any part of a triangle lies within the visual hull, the object is intersecting a real-object avatar, and a collision has occurred (Figure 1).

As in the novel viewpoint reconstruction, the algorithm first sets up *n* projected textures, one corresponding to each of the *n* cameras and using that camera's image, object-pixel map, and projection matrix. Volume-querying 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 part of the triangle being scan converted and that pixel is 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 collisions occurs if any pixel has a stencil buffer = *n*, thus indicated some part of a triangle, and in turn a virtual object, is within the visual hull.

If the triangle is projected 'on edge' during scan-conversion, the sampling of the triangle surface for volume-querying will be sparse, and collision points could be missed. No one viewpoint will be optimal for all triangles. Thus, each triangle is volume-queried in its own viewport, such 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 a view direction that is the inverse of the triangle's normal (Figure 2).

The size of the viewport and triangle affects the scan conversion sampling for volume-querying, and thus collision detection accuracy.

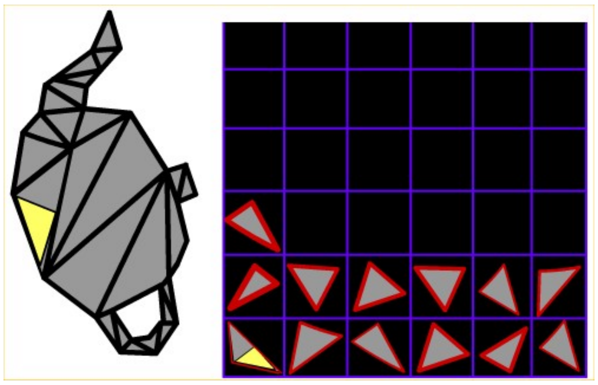

Figure 2 – Each primitive is volume queried in its own portion of the rendering window.

After all the triangles are volume-queried, 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. As an optimization, collision detection is first done with virtual object bounding boxes, and if there are collisions, on a per-triangle test is done.

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- and even object-dependent. This division of labor is similar to current collision detection algorithms [Ehmann and Lin 2000]. We provide a suite of tools to assist in moving the virtual object out of collision with the real object.

4.3. Recovery from Interpenetration

We present one approach to use the collision information to generate a plausible response. As stated before, the simplifying assumptions that make this problem tractable also make the response data approximations of the exact values. 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}, with the collision point farthest from the virtual object's reference point, RP_{obj}. The default RP_{obj} is the center of the virtual object. Due to our inability to backtrack real objects, CP_{obj} is not guaranteed to be the point of first collision of the virtual object, nor is it 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.

Our estimate to move the virtual object out of collision by the shortest distance is along a recovery vector, V_{rec} defined as the vector from CP_{obj} to RP_{obj}. This vector works well for most objects; though the simulation can specify V_{rec} for virtual objects with 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 contact on the visual hull, and to where CP_{obj} will be backed out.

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 volume-querying 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* intersects 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. *ABC* is volume-queried in the entire window's viewport, and from a viewpoint along the triangle normal and 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*, for the first pixel within the visual hull (stencil buffer value = *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 some collision detection packages [Ehmann and Lin 2000].

If the visual hull collision point surface normal, N_{hull}, is required by the application for collision response, four points on the visual hull surface near CP_{hull} are located. Stepping along *BA* and *CA* 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*. The entire process is shown in Figure 3.

Figure 5 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.

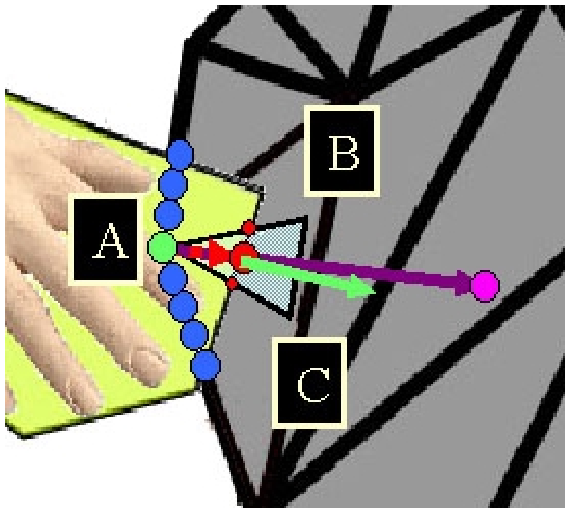

Figure 3 – CP_{obj} (A) is the farthest collision point from the virtual object's reference point (purple point). This is used to find the visual hull collision point, CPhull (red point), recovery vector (purple vector), and recovery distance (red vector).

4.4. Implementation

4.4.1. Hardware

The reconstruction algorithm has been implemented in a system that reconstructs objects within a 5' x 4' x 3' volume above a tabletop. The setup is more extensively described in [previous work]. The system used three wall-mounted NTSC cameras (720 x 486 resolution) and one camera mounted on a Virtual Research V8 HMD (640 x 480 resolution).

The participant was tracked with the UNC HiBall, a scalable wide-area optical tracker mounted on the HMD. The cameras were to an SGI Reality Monster system. The current generation of PC graphics cards 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 volume-querying, and a *simulation pipe* to run simulation and collision detection. The reconstruction was done in a 320x240 window to reduce the fill rate, and the results scaled to 640x480 – the VE rendering resolution.

The object reconstruction system runs at 15-18 FPS for a volume about 0.7 m in front of the user.

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

To speed computation, collision detection is done between the visual hull and axis-aligned bounding boxes for each of the virtual objects. For the virtual objects whose bounding box was reported as in collision with the visual hull, a per-triangle test 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, including if it should constrain the response vector. Our response algorithm then computes the recovery vector and distance. The surface normal is computed if the simulation requests that information.

4.4.2. Performance

Given n cameras, virtual objects with m triangles, and u x v viewports in a x x y window: the geometry transformation cost is ($n*m$), fill rate cost is ($n*m*u*v$)/2, and ($x*y$) pixel readbacks and compares to find collisions per frame. For collision response, the transformation cost is 2 triangles per virtual object in collision with a fill rate of ($x*y*n$) per frame.

The curtains in the window and curtain hybrid environment were composed of 720 triangles (Figure 4). The triangles were volume-queried in 10 x 10 viewports in a 400 x 400 window for collision detection, which ran at 6 frame per second.

The collision detection work was 13,000 triangles transformed per second, and 648,000 pixels per second fill rate, and 160,000 pixel readbacks and compares. The collision response work was 2 triangles and 480,000 pixels fill rate per virtual object in collision. Even though our implementation is not heavily optimized, we can achieve roughly 13,000 triangles per second for collision detection and response. As this was a first implementation, there are many optimizations that should improve the performance.

4.4.3. Accuracy

The collision detection accuracy is effected 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 [previous work]. We examine here only the effect of the volume-querying viewport and the primitive size on collision detection accuracy. The spatial sampling frequency is proportional to the size of the volume-querying viewport and inversely proportional to the number of triangles that can be queried in a single pass. The accuracy of collision detection for a u x v resolution viewport (if $u = v$, viewport layout is easier) and a triangle with x x y bounding box (in world space) is x/u by y/u. The larger the collision detection viewport, the more closely spaced the points on the triangle during volume-querying. The accuracy is the two longest dimensions of the primitive divided by the viewport horizontal size.

Primitives can be rendered at higher resolution in larger viewports, producing a higher number of more closely spaced collision points and less collision detection error. Also, a larger viewport means that fewer number of triangles can be volume queried in the collision detection window. If not all the primitives can be allocated their own viewport in a single frame buffer, then multiple volume-query and read-back cycles will needed so that all the triangles can be tested.

Hence, there is a speed-accuracy tradeoff in establishing the appropriate level of parallelism: the more viewports there are, the faster the algorithm executes but the lower the pixel resolution available for the collision calculation for each primitive. Smaller viewports have higher parallelism, but 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. The curtain system had a collision detection resolution of 0.75 cm x 0.3 cm. These values are the spatial frequency for volume-querying 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 rendering window, this results in .0125 cm error for detecting CP_{hull}. The accuracy of N_{hull}, depends on the surface topology (effected by camera resolution), the distance from these points to CP_{hull}, and the distance from CP_{hull} to CP_{obj}. The results returned by the collision response have at most a 0.75 cm error. These values are comparable to the error of the object reconstruction system, which is estimated at 0.5 cm for visual hull shape and 1.5 cm for rendering errors.

4.5. Algorithm Extensions

Figure 4 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} to the collision response algorithm). When trying to move individual cloth nodes out of collision, the motion is constrained primarily in the vector direction (V_{rec}).

Volume-querying can be done with primitives other than surface boundaries. Using primitives that represent distance fields from virtual objects would return proximity information. This 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.

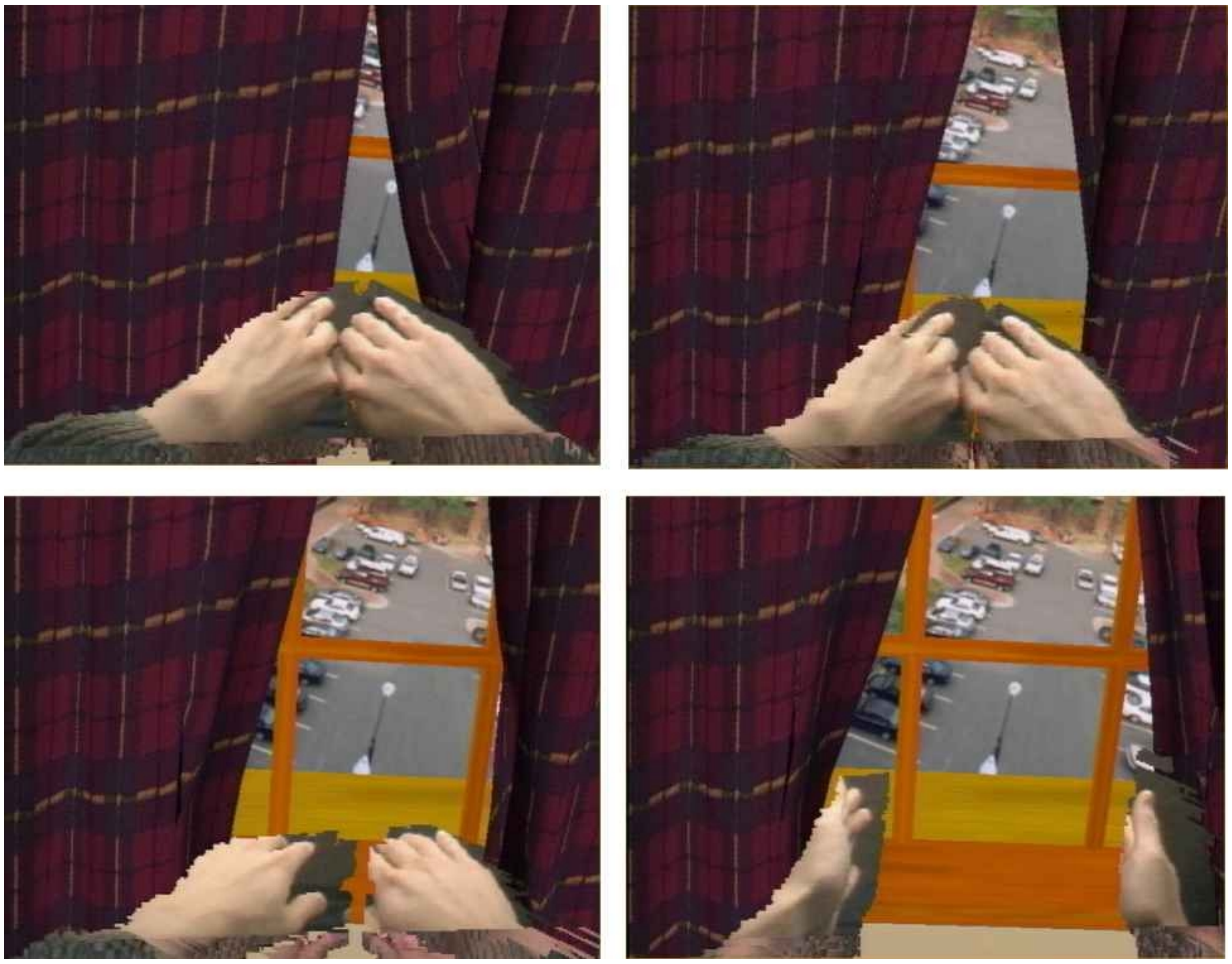

Figure 4 – Sequence of images with the user interacting with virtual curtains to look out the window.

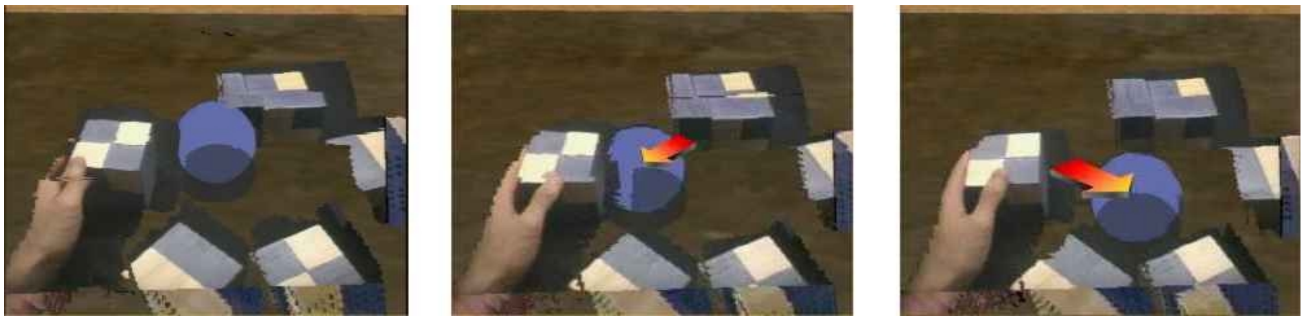

Figure 5 – A virtual ball bouncing off of real objects. The overlaid arrows shows the balls motion between images.

5. NASA Case Study

5.1. Motivation

To evaluate the potential of this technology in a real world task, we applied the reconstruction and collision detection algorithms to an assembly verification task.

We have begun a collaboration with the NASA Langley Research Center (NASA LaRC) to see how using hybrid environments could assist in evaluating payload designs and assembly layouts. Given virtual models of complex multipart devices such as satellites and engines, designers want to determine if assembling the device is physically possible. Answering this question involves managing parts, various tools, and people with a large variance in shape. Space planning errors can have a significant impact in terms of money, scheduling, and personnel. Data concerning NASA LaRC personnel motivations, comments, and suggestions are taken directly from oral or written responses to surveys, interviews, and informal remarks during experiments and discussions.

NASA LaRC designers currently 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. With the critical time constraints for payload development, testing multiple design alternatives quickly would be valuable.

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 interaction of the different subpayloads, as the complexity and nuances of each component are understood well only by a particular group. For example, attaching external cables is a common final integration task, and the NASA LaRC 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 NASA LaRC payload designers recounted several occasions when the limited fidelity of mock-ups and assembly documents caused significant problems to slip through to later stages. Payload designers also suggested that interacting with virtual models early in the design cycle would enable additional training and repetition for 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.

5.2. Payload Spacing Experiment

5.2.1. Overview

To evaluate the applicability of hybrid VEs to NASA LaRC assembly tasks, we designed an abstracted payload layout and assembly task for four LaRC payload designers. We presented the designers with task information in approximately the same manner as they receive it in actual design evaluation. They discussed approaches to the task and then executed the assembly procedure in the hybrid VE. We interviewed the designers to gather their opinions on how useful the system would be for tasks they currently have in payload assembly, testing, and integration.

During our visit to the NASA LaRC facilities, we were shown a weather imaging satellite, the CALIPSO project, and a light imager unit on the satellite called the photon multiplier tube (PMT) (Figure 6). We received CAD models of the PMT, and abstracted a task that was similar to many of the common assembly steps.

5.2.2. Assembly Task Description

The PMT model, along with two other payloads (payload A and payload B), was 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. The system indicated collisions by rendering in red the virtual object in collision.

The task, diagramed in Figure 8, was to screw a cylindrical shield (mocked-up as a PVC pipe - Figure 7) into a receptacle (Figure 9) and then plug a power connector into an outlet inside the shield (Figure 10). If the participant asked for additional assistance, we provided tools to aid in the task (Figure 11).

5.2.3. Experimental Procedure

Four NASA LaRC payload designers participated in the case study. Before attempting the task, we provided task information in approximately the same manner as they receive it in actual design evaluation. The participants were then surveyed on the space 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.

As the participant asked for more or less space, the experimenter could dynamically adjust the space between the PMT and payload A. With this interaction, different spatial configurations of the two payload subassemblies could be quickly evaluated.

5.3. Results

Given that the pipe had a length of 14 cm and a diameter of 4 cm:

Between 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 (determined 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)

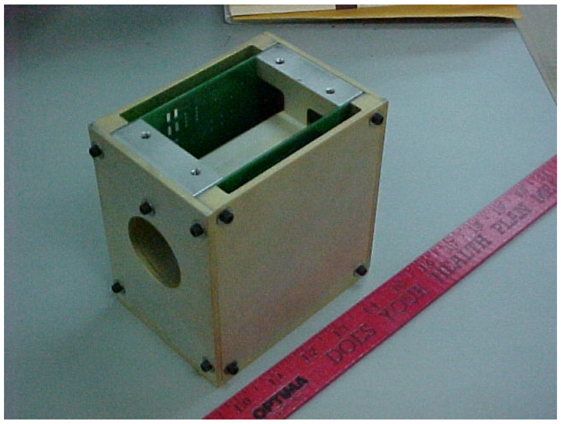

Figure 6 – Photon Multiplier Tube (PMT) box for the CALIPSO satellite payload. We used this payload subsystem as the basis for our case study.

Courtesy of NASA LaRC's CALIPSO project.

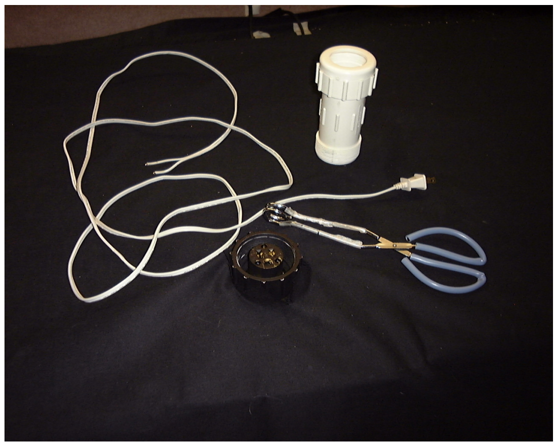

Figure 7 – Parts used in the shield fitting experiment. PVC pipe prop, power cord, tongs (tool), and the outlet and pipe connector that was registered with the virtual model.

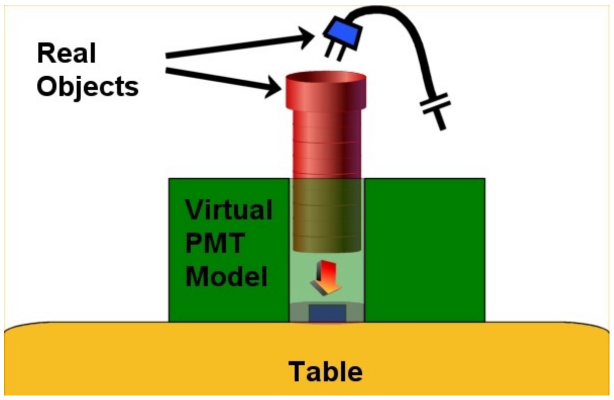

Figure 8 – Cross-section diagram of task. The pipe (red) and power cable (blue) need to be plugged into the corresponding connector down the center shaft of the virtual PMT box.

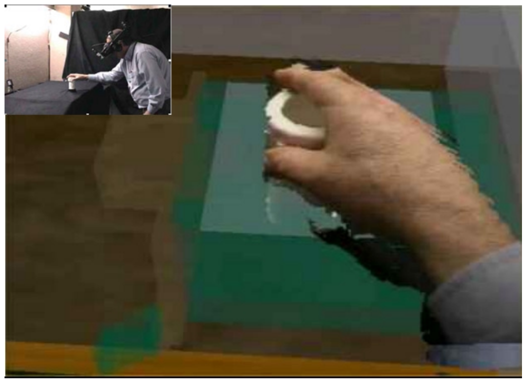

Figure 9 - The first step was to slide the pipe between the payloads and then screw it into the fixture.

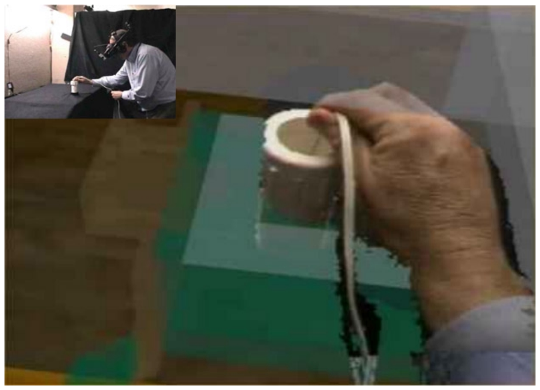

Figure 10 – After the pipe was in place, the next step was to fish the power cable down the pipe and plug it into the outlet on the table.

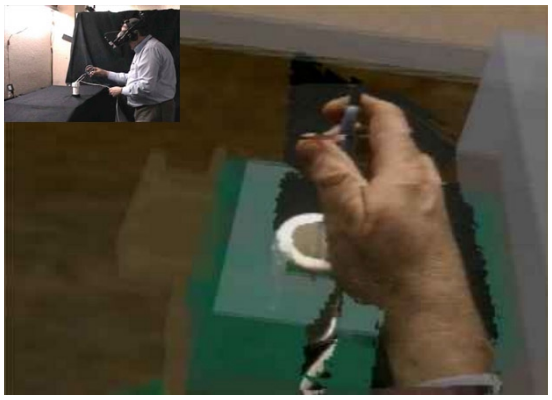

Figure 11– The insertion of the cable into the outlet was difficult without a tool. Tongs were provided to assist in the plugging in the cable.

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 possibility of requiring unforeseen 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) and were then able to complete the power cable insertion task. This required increasing the spacing between the PMT and Payload A as to avoid collisions from 14 cm to an average of 24 cm.

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 way 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 allocated 5.6 cm too little space between the payloads on their pre-experience surveys.

Accommodating tools extemporaneously, without additional modeling or development, enabled easy evaluation of multiple layouts, approaches, and tools. 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 with a final payload. These characteristics reduce their use early in the design evaluation stage. Compared to physical mock-ups, hybrid VEs can provide a cheaper and quicker alternative for evaluating designs and layouts, especially in the early design phases.

5.4. 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 commented on the potential time and schedule savings of evaluating designs at the model 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). Time is the most precious commodity in payload development, and identifying potential critical-path delays would save days to weeks.

The virtual model was not very detailed, and the visual contrast between real and virtual objects was rather obvious. Yet, participants made self-described concerted efforts to avoid touching the virtual model. Upon being told about his effort to avoid touching the purely virtual PMT box, a participant said, "that was flight hardware… you don't touch flight hardware." The familiarity and relevancy of the task made the study 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. The NASA LaRC payload designers and engineers were very optimistic about applying traditional VEs, object reconstruction VEs, and simulations to aid in payload development. They are interested in looking at 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. But, using virtual models has the most significant benefit in the design stage. Further, early identification of assembly, integration, or design issues would result in considerable savings in terms of time, money, and man-hours. Many of their tasks involve technicians interacting with a payload with tools and parts. These tasks are well suited to be simulated within an object reconstruction VE.

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 with collision detection and response would focus on improving performance, increasing accuracy, and developing algorithms to provide better collision response information.

The present limitation in our responding to collisions follows from the inability to backtrack the motions of real objects. Keeping previous camera images, along with tracking real objects within the camera images, would 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 would provide simulations with additional information 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 benefit in hardware layout and assembly verification tasks. Future work would include identifying the tasks that would most benefit from having the user handle dynamic real objects.

7. Acknowledgements

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. In *Proceedings of 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. M. Cohen and D. Zeltzer, Eds., 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*, Blackwell Publishers, 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*.

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*, ACM Press / ACM SIGGRAPH, New York, K. Akeley, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 131-144.

Lindeman, R., Sibert, J. and Hahn, J. 1999. Hand-Held Windows: Towards Effective 2D Interaction in Immersive Virtual Environments, In *IEEE Virtual Reality*.

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

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.