Online Model Reconstruction for Interactive Virtual Environments

Benjamin Lok
University of North Carolina at Chapel Hill
Department of Computer Science

ABSTRACT

We present a system for generating real-time 3D reconstructions of the user and other real objects in an immersive virtual environment (IVE) for visualization and interaction. For example, when parts of the user's body are in his field of view, our system allows him to see a visually faithful graphical representation of himself, an *avatar*. In addition, the user can grab real objects, and then see and interact with those objects in the IVE. Our system bypasses an explicit 3D modeling stage, and does not use additional tracking sensors or prior object knowledge, nor do we generate dense 3D representations of objects using computer vision techniques. We use a set of outside-looking-in cameras and a novel *visual hull* technique that leverages the tremendous recent advances in graphics hardware performance and capabilities. We accelerate the visual hull computation by using projected textures to rapidly determine which volume samples lie within the visual hull. The samples are combined to form the object reconstruction from any given viewpoint. Our system produces results at interactive rates, and because it harnesses ever-improving graphics hardware, the rates and quality should continue to improve. We further examine real-time generated models as active participants in simulations (with lighting) in IVEs, and give results using synthetic and real data.

Additional Keywords: Avatars, Visual Hull, Virtual Reality, Head Mounted Displays, Frame Buffer Tricks, HCI (Human-Computer Interface), Image-Based Rendering

CG Keywords: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual Reality; I.3.3 [Computer Graphics]: Bitmap and Framebuffer Operations, I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction Techniques

1. INTRODUCTION

We present a technique to generate view-dependent representations of dynamic real objects for rendering and natural interactions with synthetic objects in immersive virtual environments (IVE). When users move one of their arms into the field of view, we want to show an accurately lit, pigmented, and clothed arm. Research has shown that users develop a stronger sense of presence when presented with an avatar [13, 16]. In addition to increasing presence, avatars enable increased interaction, such allowing the user to affect a particle system.

Generating exact models of the user in real-time would allow the system to render a visually faithful avatar. Calculating exact models is difficult and not required for many real-time applications. A useful approximation is the visual hull. A shape from silhouette concept, the visual hull is the tightest model that can be obtained by examining only object silhouettes [5].

Our system uses graphics hardware to accelerate examining a volume for the visual hull. By using the framebuffer to compute results in a massively parallel manner, the system can generate reconstructions of real scene objects from arbitrary views in real-time. The system discretizes the 3D visual hull problem into a set of 2D problems that can be solved by the substantial yet specialized computational power of graphics hardware. The resulting dynamic representations are used for displaying visually faithful avatars and objects to the user, and as elements for interactions with VEs and simulations. As the capabilities and performance of graphics hardware continue to improve, our system will increase in accuracy, improve reconstruction quality, and improve the interactions between real and synthetic objects.

1.1 RELATED WORK

There are several approaches to the difficult problem of recovering accurate three-dimensional models from real-world images. The Virtualized Reality project by Kanade [4] and the telepresence work by Sara and Bajcsy use dense stereo algorithms [11]. Segmentation and space carving algorithms partition volumes using range and/or camera data [1,2,9]. Computer vision techniques attempt to understand the content of images so as to find correlations for extracting models and environments [3,8,10].

At SIGGRAPH 2000, Matusik, etal., presented an image-based visual hull algorithm, “Image Based Visual Hulls” (IBVH), that used clever traversing of pixels between camera images to reconstruct models [6]. IBVH uses multiple cameras and software based IBR techniques to compute the visual hull at interactive rates. While IBVHs are well suited for novel viewpoint scene reconstruction, our technique is not sensitive to the percent of the viewport covered by the reconstruction. The multiple reconstructions per frame used for avatars, dynamic models in simulations, and interaction techniques usually fill the viewport. Our algorithm’s approach of “querying whether points are within the visual hull” allows incorporation of real items as active objects of dynamic simulations. For example, collision detection can be done through rendering the synthetic objects and determining intersections with the visual hull of real objects. Our algorithm inherently uses graphics hardware to identify intersections. System performance is not tied to reconstruction complexity and benefits from the speed and capability improvements that accompany new generations of graphics cards.

Research on avatars in virtual reality has mostly focused on sense the presence and appearance. Most existing VE systems, such as the one used in the Walking > Virtual Walking > Flying, in Virtual Environments project by Usoh et al., required additional trackers to control the motion of a stock avatar model [16].

Interaction with objects in VEs is accomplished by translating hardware actions, such as button pushes, to actions such as grasping. Existing hardware works for some tasks, but for others; the mapping is unnatural. For example, to pick up a book a user moves the avatar hand to intersect the virtual book, and then presses and holds the trigger. Our system enables users to see their arm and a real book in the VE, and naturally reach out and pick up the book.

Our system has two distinct goals, providing the user visual feedback of his body and nearby real objects, and generating real-time models for rendering, simulations, and interactions with the virtual environment. We discuss each in turn.

2. 3D RECONSTRUCTION

The reconstruction algorithm has four steps: image processing to label object pixels, calculating the volume intersection, rendering the visual hull, and compositing with the virtual environment.

2.1 OBJECT PIXELS

Fixed cameras are positioned with each camera's frustum completely containing the volume to be reconstructed. Except for the user and objects of interest, the real-world scene is assumed to be primarily static. We use image subtraction to extract pixels of interest from the background. When initializing, the system captures reference images of the vacated scene for each camera. For subsequent frames, the reference image is subtracted from the current image. The results are compared against a threshold to determine if the image pixel represents part of a new scene object [14]. We label pixels with differences below the threshold, **background pixels,** and the rest, **object pixels**. After startup, only newly introduced objects appear as object pixels because their color should substantially differ from the reference pixels. Although image subtraction can return slightly noisy results, it produces results that are efficient to compute, easy to load balance across multiple processors, and sufficient for reconstruction.

2.2 VOLUME INTERSECTION

For some applications not requiring precise models, the visual hull is an acceptable representation. The visual hull can be obtained by determining the occupied volume.

A new object in the scene appears as object pixels on *every* camera's image. If we reverse the idea, object pixels on a camera's image represent the presence of an object within a pyramidal volume swept out from the camera's center of projection, through the object pixel, and into the scene. Collectively, the background pixels in a camera's image represent a projection mask. The rays that pass through this mask form a volume that *could* contain objects. The object pixels carve out a volume in which objects *potentially* exist. The visual hull is the 3D intersection of all **object pixel projection volumes**.

Volume visualization algorithms usually build a structure of voxels from images and then traverse it to generate a novel view [15]. A significant amount of work can be avoided by only examining parts of the volume that could contribute to the final image. Instead of trying to compute entire volumes or the intersections of potentially complex polygons that represent the object pixel projection volumes, we ask, "which points in the view volume are inside the visual hull?"

We use synthetic data of a 3D model rendered from five known locations to illustrate our reconstruction technique [Figure 1]. To determine if a 3D point is within the visual hull, it must project onto an object pixel in *every* camera's image, and thus be within the object pixel projection volume. To perform the intersection tests between the projection volumes is too expensive for even a modest setup. Numerical stability and robustness problems aside, scalability becomes an issue as the number of intersection tests grows rapidly with the number of cameras. Even if simplified with hierarchical or algorithmic methods, the brute force approach challenges the computation and bandwidth capabilities of most machines. We use the massively parallel computational power of graphics hardware to compute a view specific pixel-accurate volume intersection in real-time.

2.3 HARDWARE ACCELERATION

We use projected textures and the framebuffer to perform intersection tests in parallel, and compute only the elements that could be contained in the final image.

A single point: To determine if a 3D point is in the visual hull, we render it n times (where n is the number of cameras). When rendering the point for the ith time, the texture matrix stack is set to the projection * modelview matrix defined by camera i's extrinsic parameters. This generates texture coordinates that are a perspective projection of image coordinates from the camera's location. The camera's image is converted into a texture with the alpha of object pixels set to 1 and background pixels set to 0. An alpha test to render only texels with alpha=1 is enabled. If the point is textured, it projected onto an object pixel in camera i. If all n cameras project an object pixel onto the point, then the point is within the visual hull [Figure 2]. The system uses the stencil buffer to accumulate the number of cameras that project an object pixel onto the point. The pixel's stencil buffer value is incremented for each camera that projects an object pixel onto the point. Once all n textures are projected, we change the stencil test and redraw the point. If the pixel's stencil buffer value $<n$, it means there exists at least one camera where the point did not project onto an object pixel and is not within the visual hull. The stencil and color elements for that pixel are cleared [Figure 3]. If the pixel's stencil buffer value equals n, it is within the visual hull. In effect, we are querying the visual hull for an intersection with whatever primitive we render.

In implementation, we ignored the cameras' intrinsic parameters, and future work into camera calibration is planned. However, we have found only using the extrinsic parameters does not greatly reduce the quality of the reconstruction for small volumes.

Parallelization: For all points within the reconstruction view frustum, we want to query for inclusion within the visual hull. To approximate this, we sample the volume by rendering planes and make use of the framebuffer's parallel nature to resolve occlusion. To generate the first visible surface, a set of planes, orthonormal to the view direction and completely filling the rendering window, are drawn from front to back. This is similar to other plane sweep techniques for combining textures [7]. For each plane, an optimization is to compute a bounding box for each camera's object pixels and project them onto the plane. The intersection of the box projections reduces the size of the plane we render. To compute the projection volume intersections, the plane is drawn $n+1$ times, once with each camera's projected texture, and once to keep only pixels with a stencil buffer value = n. Pixels with a

stencil value of *n* represent points on the plane that are within the visual hull. When drawing the planes from front to back, the color and stencil buffers are not cleared between planes. The result is a correctly z-buffered first visible surface of the visual hull from the user's viewpoint [Figure 4]. The number and spacing (uniform or non-uniform) of the planes are user defined and depends the complexity of objects, reconstruction resolution, and application requirements.

For *n* cameras, with *u* x *v* pixel image planes, rendering *p* planes requires (2*n*+2)*p* triangles. When rendering each plane, we are computing *uv* volume intersection results. Current graphics hardware is capable of drawing between 10^7 – 10^9 textured triangles per second. For large volumes with several cameras, our geometry requirements are less than 10^6 triangles per second.

The two bottlenecks of the algorithm are fill rate and image subtraction. Since every pixel on the screen is rendered *n*+1 times per plane, *fill rate* = (*n*+1)**p***u***v*. Current graphics hardware have fill rates between 10^8 – 10^9 pixels per second and this restricts the reconstruction quality and resolution. The other computationally expensive component of the algorithm is obtaining the object pixels per camera. Since each pixel has to be subtracted each frame, the number of subtractions required is *u***v***n*. Even with a few cameras, this is expensive to compute each frame. With image processing hardware becoming more readily available, we look to integrate accelerated image subtraction results. The geometry, fill rate, and computation requirements scale linearly with the number of cameras, and the results depend substantially on camera resolution, object complexity, and number of cameras and planes. We are employing this system on desktop tasks that will make use of real-time approximate models of real objects within a volume.

2.4 COLORING THE MODEL

There have been several methods proposed to use the source camera images for reconstructing and texturing a model [12]. We compute the final color by using the image from the HMD mounted camera to texture the reconstruction result.

If rendering other than from the HMD view is required, then color data from the source cameras are used. Since our algorithm does not build a traditional model, computing visibility per pixel is expensive. The "Image Based Visual Hulls" algorithm by Matusik computes both the model and visibility by keeping track of which source images contribute to a final pixel result.

To compute color using source images, we generate a coarse mesh of the depth buffer of the reconstruction. We assume the camera that most likely contributed to a point's color shares a view direction closest to the mesh's normal. For each mesh point, its normal is compared to the viewing directions of the cameras. Each vertex gets its color from the camera whose viewing direction most closely matches its normal [Figure 5].

3. DYNAMIC ENVIRONMENT

We sample the visual hull from the viewpoint of the user. The reconstruction is used as a surface for texturing the image from the HMD camera. Since the reconstruction is done in eye coordinates, the rendered virtual environment is composited with the reconstructed real objects [Figure 6].

Currently, interacting with virtual environments forces a mapping of virtual actions to real hardware such as joysticks or mice. For some tasks these associations work well, but for some interactions users end up fighting the affordance mismatch of the I/O device. For example, in the Walking > Virtual Walking > Flying, in Virtual Environments project, the user is instructed to pick up a book from a chair and move it around the VE. The user carries a magnetically tracked joystick, and must make the avatar model intersect the book to select it. The user then presses and holds the trigger to pick up and carry the book.

Real-time models of the user enable more natural interactions with elements in virtual environments. In the example system, we would replace the synthetic book with a real book that would be reconstructed along with the user. The user could then see the book on the chair and naturally reach out to pick it up. Additionally, the user gets the benefits of visually faithful avatars and haptic response. These new hybrid realities interact with, and are affected by, the user and real objects.

Real-time dynamic models begin to blur the line between real and synthetic objects in the VE. One important facet of this interplay is lighting and shading. We want the dynamic models of real objects to be lit by virtual lights. To do this, we enable the VE's lights while rendering a mesh of the reconstruction depth buffer. The resulting lit vertices are modulated with the applied texture. *Synthetic* lights are affecting *real* objects. Conversely, we can use traditional shadowing algorithms to cause synthetic elements in the environment to cast shadows on the real objects.

Because the reconstruction algorithm is not the bottleneck (image subtraction and data transfer are where most of the time is spent), we can use the same camera frames for reconstructions from different viewpoints without a large performance hit. Shadows can be calculated by reconstructing the scene from the viewpoint of the light, and the resulting image can be used as a shadow texture onto scene geometry. *Real* objects are affecting the *synthetic* scene. Our observations note that users show increased spatial perception (how high an object or their hand is above a table) through the presence of shadows.

Beyond just an overlay on a computer-generated scene, the dynamic models of the user *and* nearby objects can be active elements in simulations. We have implemented a particle system that represents water flow from a faucet. By reconstructing the model from above, a reconstruction of the user's hand and a newly introduced object, a plate, was used as a surface within the particle system. The water particles could interact with the plate and flow into the sink [Figure 7]. For other simulations, we can reverse the question and ask, "is this synthetic object within the visual hull?" There is no reason to restricting our sampling of the visual hull to only planes. We are working on collision detection of potentially complex synthetic models with real objects. Instead of sweeping planes, we query the visual hull for intersections by rendering the synthetic models with the projected textures and searching the framebuffer for intersections.

There are many areas that could use online reconstructed models, from exploring crowd control and robot motion planning to middle school students studying magnetism. Our real-time model reconstruction system is an enabling technology that allows virtual environments to be truly dynamic and interact with the user in completely new ways. We anticipate evaluating this system for interactions virtual characters, new navigation methods, and its affect on presence in VR.

4. IMPLEMENTATION

We have incorporated the reconstruction algorithm into a system that reconstructs a 8-ft x 6-ft x 6-ft volume. The system uses five wall-mounted NTSC cameras (720x486) and one camera mounted on a Virtual Research V8 HMD (640 x 480). The user is tracked with a scalable wide-area optical tracker [18].

When started, the system captures and averages a series of images for each camera for the background "reference" images. Since NTSC divides each frame into two fields, two reference images are stored per camera. Reconstruction is done per field. While this increases the error, latency is reduced and dynamic objects exhibit less shearing. The quality of the visual hull fit is directly related to and restricted by the resolution of the camera images.

The six camera inputs are connected to an SGI Reality Monster system. While PC graphics cards could handle the graphics requirements of the algorithm, the SGI's ability to simultaneously acquire multiple color-camera images and its high memory bandwidth made it a better solution. As PCs handle more digital video and memory and system bus bandwidths improve, they could become a viable platform for the reconstruction technique.

We use one parent and three child pipes to parallelize the reconstruction. The number of child pipes is a trade off between latency and frame rate, both of which increase with more pipes. The parent pipe obtains and broadcasts the camera images. In frame sequential order, the child pipes do the image subtraction, reconstructs the model, and transfers the results. The results are then combined with the virtual environment.

Four processors are used to perform the image subtraction, and the reconstruction volume is sampled at centimeter resolution for 2 meters in front of the user. Rendering the results into a 320x240 window (scaled for the HMD display), the system achieves 12-15 frames per second. The graphics complexity of the reconstruction is ~45,000 triangles per second with a $1.7 * 10^9$ pixels per second fill rate. The latency is about 0.3 of a second.

Our VE of a room with a faucet and sink uses the real-time model reconstruction technique. The user has a visually faithful avatar that casts shadows onto the environment and interacts with a water particle system from the faucet. We observed users cup their hands to "catch" the water, put random objects under stream to watch particles flow down the sides, and comically try to drink the synthetic water. Unencumbered by additional trackers and intuitively interacting with the virtual environment, users exhibit an approach of uninhibited exploration, often doing things "we didn't think about."

5. RESULTS AND FUTURE WORK

We have presented an algorithm that generates a real-time view dependent sampling of the visual hull by using graphics hardware acceleration. The resulting reconstructions are used to generate visually faithful avatars and as active objects in IVEs. The system does not require additional trackers or require *a priori* objects information, and allows for natural interaction between the objects and virtual environment.

Our system development will focus on collision detection, improved rendering techniques, and exploring applications. The algorithm is being refined to better compensate for image subtraction and camera calibration error. We plan on a user study to examine the effects on task performance of visually faithful avatars and natural interactions with the VE. We assert that the unencumbered interactivity and improved immersion will enhance exploration and task performance. Through expanding the interactions between real objects and the synthetic environment, we seek to enable a new type of hybrid reality.

6. ACKNOWLEDGEMENTS

Supported in part by an NSF Fellowship. I would also like to thank Mary Whitton, Greg Welch, and Samir Naik for extensive help with editing, and the reviewers for their helpful comments.

7. REFERENCES

[1] J. Carr, W. Fright, A. Gee, R. Prager and K. Dalton. 3D Shape Reconstruction using Volume Intersection Techniques. In *IEEE International. Conference on Computer Vision Proceedings*, 1095-1110, January 1998.

[2] C. Chien and J. Aggarwal. Volume/Surface Octrees for the Representation of Three-Dimensional Objects. *Computer Vision, Graphics, and Image Processing*, volume 36, No. 1, 100-113, October 1986.

[3] I. Kakadiaris and D Metaxas. Three-Dimensional Human Body Model Acquisition from Multiple Views. *Int'l Journal of Computer Vision 30*, 1998.

[4] T. Kanade, P. Rander , S. Vedula and H. Saito Virtualized Reality: Digitizing a 3D Time-Varying Event As Is and in Real Time. In Yuichi Ohta and Hideyuki Tamura, editors, *Mixed Reality, Merging Real and Virtual Worlds*. Springer-Verlag, 41-57, 1999.

[5] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 16, No. 2, 150-162, February 1994.

[6] W. Matusik, C. Buehler, R. Raskar, S. Gortler and L. McMillan. Image-Based Visual Hulls. In *SIGGRAPH 00 Proceedings*, Annual Conference Series, pages 369-374.

[7] A. Meyer and N. Fabrice. Interactive Volumetric Textures. *Eurographics Workshop on Rendering.* June 1998.

[8] S. Moezzi, Katkere, A., Kuramura, D. Y., & Jain, R. Reality Modeling and Visualization from Multiple Video Sequences. *IEEE Computer Graphics and Applications*, 16(6), 58-63.

[9] M. Potmesil, Generating Octree Models of 3D Objects from Their Silhouettes in a Sequence of Images. *Computer Vision, Graphics and Image Processing*. Vol 40, 1-29, 1987.

[10] J. Rehg and T. Kanade. Digiteyes: Vision-based hand tracking for human-computer interaction. In J. Aggarwal and T. Huang, editors, *Proceedings of Workshop on Motion of Non-Rigid and Articulated Objects*, pages 16-22, IEEE Computer Society Press. November 1994.

[11] R. Sara and Ruzena Bajcsy. Fish-Scales: Representing Fuzzy Manifolds. In *Proceedings of IEEE Conference ICCV '98*, pages 811-817, January 1998.

[12] S. Seitz. and Dyer C. Photorealistic Scene Reconstruction by Voxel Coloring. *Computer Vision and Pattern Recognition Conference*, 1067-1073, 1997.

[13] M. Slater and Martin Usoh. Body Centred Interaction in Immersive Virtual Environments, in N. Magnenat Thalmann and D. Thalmann, editors, *Artificial Life and Virtual Reality*, pages 125-148, John Wiley and Sons, 1994.

[14] A. Smith and J. Blinn. Blue Screen Matting. *Computer Graphics (SIGGRAPH 96 Conference Proceedings)*, pages 21-30, August 4-9 1996.

[15] R. Szeliski. Rapid Octree Construction from Image Sequences. *Computer Vision, Graphics, and Image Processing: Image Understanding*, pages 23-32, Vol. 58, No. 1, July 1993.

[16] M. Usoh, K. Arthur, et. al. Walking>Virtual Walking>Flying, in Virtual Environments. *Proceedings of SIGGRAPH 99*, pages 359-364, Computer Graphics Annual Conference Series, 1999.

[17] S. Vedula, P.W. Rander, H. Saito H., and Takeo Kanade. Modeling, Combining, and Rendering Dynamic Real-World Events From Image Sequences. *Proceedings of the 4th Conference on Virtual Systems and Multimedia*, Vol.1, pages 326-332, November 1998.

[18] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, D. Colucci. The HiBall Tracker: High-Performance Wide-Area Tracking for Virtual and Augmented Environments. *Proceedngs of the ACM Symposium on Virtual Reality Software and Technology 1999*, December 1999.

Figure 1: Synthetic data set of a 3d model rendered from five positions. This is the input to the reconstruction algorithm. Our goal is to generate an arbitrary view of the user in real-time for avatars and interacting with the virtual environment.

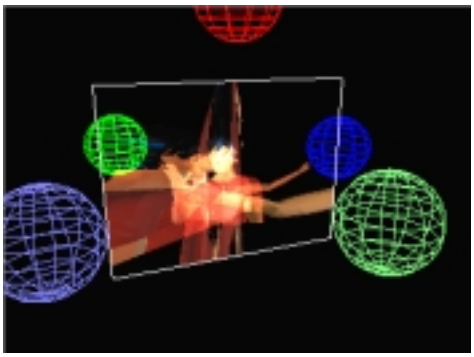

Figure 2: Using projected textures, each camera (the spheres) image is projected onto a plane.

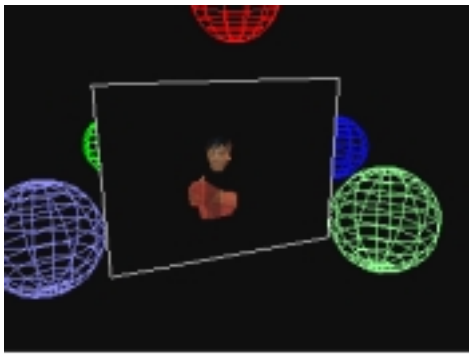

Figure 3: The points on the plane within the intersection of the texture projections are within the visual hull.

Figure 4: Compositing the planes results in a first visible surface of the visual hull. The depth buffer result is shown.

Figure 5: We dynamically reconstruct and light the model from a novel viewpoint.

Figure 6: The resulting models can be used for lighting and inputs into simulations such as collision detection.

Figure 7: With real-time approximate models of real objects they become active participants in a hybrid reality.