Visual Hull Rendering with Multi-view Stereo Refinement

Yang Liu[1] yngliu@ucdavis.edu
George Chen[2] george-qian.chen@st.com
Nelson Max[1] max2@llnl.gov
Christian Hofsetz[1] chofsetz@ucdavis.edu
Peter McGuinness[2] peter.mcguinness@st.com

[1]Center for Image Processing and Integrated Computing
2343 Academic Surge
University of California, Davis
1 Shields Ave.
Davis, CA 95616

[2]AST La Jolla Lab STMicroelectronics
4690 Executive Dr.
San Diego, CA 92121

ABSTRACT

We present a system for rendering novel viewpoints from a set of calibrated and silhouette-segmented images using the visual hull together with multi-view stereo. The visual hull predicted from the object silhouettes is used to restrict the search range of the multi-view stereo. This reduces redundant computation and the possibility of incorrect matches. Unlike previous visual hull approaches, we do not need to recover a polyhedral model. Instead, the visual hull is implicitly described by the silhouette images and synthesized using their projections onto a set of planes. This representation allows an efficient implementation on current pixel-shader graphics cards, yielding frames at interactive rates. We also introduce a library of image filters to improve rendering results along edges and silhouette profiles.

Keywords

Visual Hull, Multi-View Stereo, Fragment Shader

1. INTRODUCTION

Given a set of calibrated and silhouette-segmented images, the visual hull [Laur94a, Mat01a, Mat00a] is the maximal solid volume that is consistent with each silhouette. The visual hull is given by the intersection of the input silhouette cones. The resulting intersection volume is a conservative bound on the object's shape. As more images become available, the visual hull will converge to a volume that is tighter than the object's convex hull, but this volume will not necessarily converge to the true geometry. This is due to the potential presence of concave regions on the object that are difficult, if not impossible, to detect using only silhouettes. Despite this shortcoming, the visual hull is still a good first approximation to the actual object geometry.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

Multi-view stereo reconstruction algorithms, in contrast to shape-from-silhouette methods, are able to recover these concave regions (as long as sufficient texture information is present) but typically fail to accurately recover geometry along the silhouette profiles. However, multi-view stereo can take advantage of the visual hull to avoid unnecessary computations for locations outside of the object volume (this can potentially reduce the possibility of incorrect matches), which can help to recover the object profile. The stereo, which relies on image texture, can also recover concave regions that are invisible to the visual hull. The two methods are thus complementary.

Li, et al. proposed a method [Li03a] to recover the full polyhedral visual hull model by computing constructive solid geometry (CSG) intersections in graphics hardware. The acquired model was then used to restrict the per-pixel epipolar search range for multi-view stereo reconstruction. However, this approach requires significant computational effort to process dynamic scenes; recovering the polyhedral model at interactive rates, despite hardware acceleration, is still prohibitively expensive. Furthermore, a significant portion of the algorithm was implemented outside of the graphics card.

Since we are interested only in rendering the visual hull, it may not be necessary to recover the polyhedral model. Li later explored this idea and presented a multi-pass technique [Li03b] for real-time rendering of visual hulls using hardware stencil buffers. In this approach, Li eschewed a polyhedral visual hull for a view-dependent depth map representation, which was then projectively texture mapped using the appropriate set of input images for image synthesis. However, the results demonstrate that the visual hull approximation to the geometry is not sufficient for rendering convincing images. Blurring and ghosting artifacts were present in the images since the visual hull failed to accurately represent the object geometry. As expected, the artifacts were especially noticeable for concave elements of the object surface.

Yang, et al. [Yan00a, Yan02a] introduced a multi-pass approach to stereo reconstruction that was implemented using programmable graphics hardware. The results are superior to those of the visual hull, but there are still observable artifacts present in the images, especially along the object edge profiles, which could have been removed by considering the visual hull. Furthermore, his implementation on earlier graphics hardware required expensive buffer "read-backs" from the texture memory to perform the matching score aggregation, which further penalized the frame rate.

We present in this paper a new multi-pass plane-sweep rendering method that takes advantage of the visual hull and multi-view stereo reconstruction together to synthesize images from new viewpoints. The approach we describe here is, in spirit, an integration of the two previous techniques reported by Li and Yang, with additional image filters to improve the rendering quality. We outline some of our principal contributions:

- We use the visual hull implied from the image silhouettes to restrict the search range for each pixel in the multi-view stereo. This allows us to avoid unnecessary computations and reduce potential false matches.
- Our method uses a library of template shapes to identify correspondences along surface discontinuities at the visual hull profiles. Our results demonstrate how the templates improve the rendering results in the presence of these discontinuities.
- The hardware implementation does not require any "read-backs" from texture memory. All rendering operations are performed within the graphics card and intermediate rendering states are written to texture memory via off-screen buffers.

The remainder of this paper is organized as follows. We overview our rendering system in the next section and present our algorithm in section 3 and discuss details of its hardware implementation in section 4. We present results in section 5, and conclude with ideas for future work.

2. RENDERING SYSTEM

Our system consists of 6 calibrated DragonFly Firewire cameras observing a dynamic environment. We define a spline path through the 6 cameras and reconstruct images using the closest 4 cameras for each given virtual intermediate viewpoint. Images from this set of four cameras are then acquired and processed by fragment programs on an ATI Radeon 9800 graphics card. These fragment programs implement the visual hull and multi-view stereo reconstruction, which we describe later.

The fragment programs are invoked in the graphics hardware by initializing a set of sweeping planes. Each of these planes induces a mapping transformation, which we use to transfer pixels across images. The planes may be processed independently and are composited into a color and depth buffer for direct display. The rendering rate trades off with the reconstruction quality by adjusting the number of planes to process in the multi-view stereo.

3. SYSTEM OVERVIEW

To synthesize an output view for each new viewpoint, the algorithm basically steps a sample plane parallel to the output image plane through the scene depth range, choosing for each output pixel the depth of the best match, as computed from the multi-view stereo. By considering a set of sample planes through this range, we may recover scene geometry by selecting from each of the planes, match locations with high color consensus. To recover a depth map from the desired viewpoint, we intersect rays through the set of planes and select for each depth pixel, the location with the best matching score from the set of plane intersections.

We proceed by first preparing a bit-mask image to describe the segmentation of the foreground object silhouette against the static background. The input images along with their respective bit masks are mapped into the virtual view by a plane-induced transformation. We then determine locations on the sample plane that lie within the object's visual hull by comparing the mapped bit-mask images; locations outside of the visual hull are identified and discarded from further processing.

```
segment silhouettes from input images
for each plane
    map input  images onto plane
    compute visual hull intersection of plane
    compute color consensus and mean color
    mask color consensus image by visual hull
    aggregate match scores (using template library)
    render mean colors into color buffer and
        matching scores into Z-Buffer
end for
display color buffer
```

Figure 1. Outline of rendering algorithm.

For the multi-view stereo, we map the input images, using a mapping transformation defined by each sample plane, to compute the mean color and the color consensus of coincident pixels (the set of pixels that map to the same location.) We assume that the coincident pixels on or near locations where the sample plane intersects the object surface will have high color consensus (low color variance), while coincident pixels off of the surface will have low consensus (high color variance). We use the converse of this principle to infer matches – locations on the plane that are good estimates to the object geometry (i.e. plane-object intersections.) However, the color consensus is not sufficient to identify matches on image regions that are untextured or contain repeated patterns. This is a weakness that is common to most stereo algorithms that rely on color texture to assess geometry.

We refine our initial consensus estimation by examining a local neighborhood of pixels to aggregate support for each matching location. Deciding which pixels to include in this neighborhood is critical for establishing a match. This is especially important for matches that are located along surface discontinuities where the local neighborhood may have a large discrepancy in matching scores. We address this problem by searching through a library of template shapes to find the correct neighborhood for each pixel.

We prepare output images by rendering each plane, writing its depth and matching scores into the respective color and Z buffers. For direct view synthesis, we substitute the depth with the mean color. We can also directly display the color consensus score to visualize the confidence of our matches. The minimum & maximum depths and number of planes control respectively the search range and sampling resolution of the multi-view stereo. We present an outline of the algorithm in Figure 1.

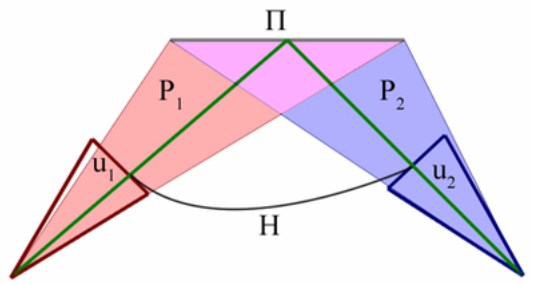


Figure 2. *H* is a homography induced by plane *Π* and camera projection matrices P_1 and P_2. We use *H* to map pixels u_1 and u_2 between the two camera views.

4. HARDWARE IMPLEMENTATION

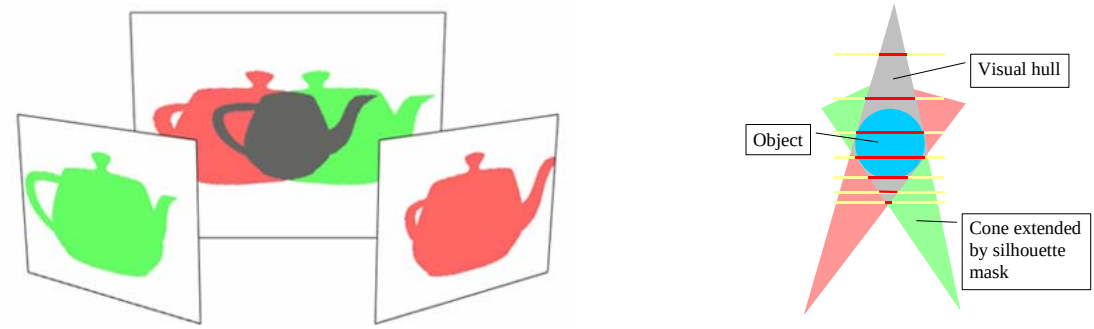
We implement our rendering algorithm on the ATI Radeon 9800 Pro, taking advantage of its programmable pixel shader units through judicious use of fragment programs [Frag03]. Each program execution requires a separate rendering pass. The resulting image from each intermediate pass is rendered off-screen into a texture buffer and then bound as texture input to a fragment program for the next rendering pass. By storing the intermediate results inside of the texture memory, we avoid expensive transactions across the system bus.

Our algorithm processes a set of sample planes that sweeps out a bounding volume of the object. Each plane requires several rendering passes to fully process. For each plane, we compute the average color and matching score over all locations on the plane. The planes are independent of each other and may be processed concurrently and independently, but currently, the hardware pipeline restricts us to a sequential processing order.

In this section, we describe how we implement each stage of our algorithm in graphics hardware. The silhouette segmentation is performed once for each set of input images. All other processing stages, such as the plane-mapping, visual hull/color consensus, and aggregation steps are performed separately for each new output view, and each sample plane.

Silhouette Segmentation

We prepare the input images by first segmenting them to find the object silhouettes using standard background subtraction. We sample an initial image, without the object to be segmented and denote this the "background image." Each subsequent image is then subtracted against the background image. We use a threshold to identify the foreground pixels and encode the result as a bit-mask image. If the foreground object contains colors similar color to the background then the bit-mask image may contain



(a) **(b)**

Figure 3: (a) Two silhouette bit-masks are projectively mapped onto a plane. The intersection of the silhouette projections (shown in gray) is a cross-section of the visual hull. Additional silhouette bit-masks from new viewpoints will further constrain the intersection region and the visual hull. (b) When the cross-sections are collected together, the visual hull volume is discretized to the set of planes. We can display the visual hull by directly rendering the cross-sections from each plane.

holes. We can correct this problem using median or morphological filters (such as the closing operator) to remove the holes. The final bit-masks are stored into the alpha channel of each respective input image. We perform this segmentation for all of the acquired images.

Plane-Mapping

We use plane-to-plane mappings – or homographies – to facilitate the transfer of pixels between images. Each homography is linear in projective space and may be described by a 4x4 matrix:

$$\begin{aligned} \mathbf{u}_1 &= \begin{bmatrix} u_1 & v_1 & 0 & 1 \end{bmatrix}^T \\ \mathbf{u}_2 &= \begin{bmatrix} u_2 & v_2 & 0 & 1 \end{bmatrix}^T \\ \mathbf{u}_1 &= H \cdot \mathbf{u}_2 \end{aligned} \qquad (1)$$

(u_1, v_1) and (u_2, v_2) are pixel locations in the two images, and H is a homography defined between them (Figure 2), induced by the plane Π. It can be constructed from the camera projection matrices P_1 and P_2 for two cameras and the equation of plane Π as follows:

$$\Pi = \begin{bmatrix} a & b & c & d \end{bmatrix}$$

$$P_i = \begin{bmatrix} p^i_{00} & p^i_{10} & p^i_{20} & p^i_{30} \\ p^i_{01} & p^i_{11} & p^i_{21} & p^i_{31} \\ p^i_{02} & p^i_{12} & p^i_{22} & p^i_{32} \end{bmatrix}$$

$$\overline{P}_i = \begin{bmatrix} p^i_{00} & p^i_{10} & p^i_{20} & p^i_{30} \\ p^i_{01} & p^i_{11} & p^i_{21} & p^i_{31} \\ a^i & b^i & c^i & d^i \\ p^i_{02} & p^i_{12} & p^i_{22} & p^i_{32} \end{bmatrix} \qquad (2)$$

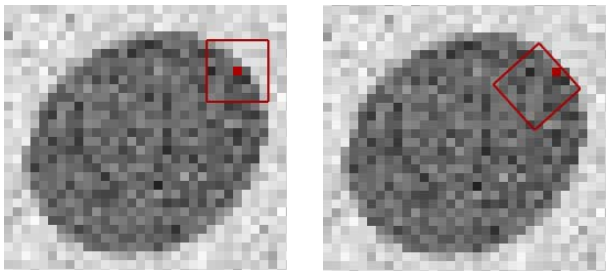
$$H = \overline{P}_1 \overline{P}_2^{-1}$$

where, for i = {1,2}, $\overline{P}_i$ is a 4 x 4 matrix created from P_i by inserting the plane coefficients $\begin{pmatrix} a^i & b^i & c^i & d^i \end{pmatrix}$ (transformed to view i's coordinate frame) into the third row of the matrix. For our system, we use planes that are parallel to the virtual image plane. We rely on these transformations to map pixels to the virtual viewpoint (P_1) from an input image (P_2).

We set up a homography in OpenGL by initializing a plane that occupies the entire viewing region. The texture matrices are then set with the appropriate homography coefficients and we load each texture unit with its respective input image. As the plane is rasterized, the input images are projectively textured onto it. Each screen pixel now contains the set of coincident input pixels and bit-masks. The coincident input pixels may be used to determine how well they match at that location. Similarly, the coincident bit masks may be used to decide if that location is within the object visual hull.

Visual Hull

After the input bit-mask images are projected (and resampled using nearest neighbor interpolation) onto the plane, we use them to determine locations on that plane that are within the visual hull by computing the logical AND (i.e. intersection) for each set of coincident bit masks. Thus, if at least one camera does not see a location, then that location is labeled as outside of the visual hull. The result of this operation is a binary image, which describes the intersection of the plane with the visual hull deduced from the object silhouettes. We illustrate this idea for a pair of 2D cameras in Figure 3a. This step does not require its own rendering pass; we combine it together with the color consensus computation into a single fragment program. The full visual hull can be



(a) **(b)**

Figure 4: The matching score image is typically noisy. We use a template window to consider the local pixel neighborhood to support each match and remove the noise. The template shown in (a) lies across a surface discontinuity. Even though the pixel is on the object surface and should be correctly matched, its neighborhood contains pixels that are off of the surface whose high scores will inflate the aggregated matching score. The template shown in (b) has a pixel neighborhood that does not contain any discontinuities and will not be misled by nearby false matches.

recovered in this manner by sweeping the plane through an initial bounding volume of the object such that the visual hull is discretized to a set of planes (Figure 3b). We can directly render the visual hull in this representation by directly rendering the planes.

Color Consensus

We use the color consensus of coincident input pixels to estimate the likelihood of a match, that is, each pixel is a projection of the same location – a correspondence. We compute the statistical variance of the color to measure the color consensus for a set of pixels. For a set of pixel colors, $\{c_i \mid i \in 1,2,...,N\}$, ($N$ = 4, in our case), the variance (c_{var}) is given by the following computations:

$$\bar{c} = \frac{1}{N}\sum_i^N c_i$$
$$c_{\text{var}} = \frac{1}{N}\sum_i^N (c_i - \bar{c})^2 \qquad (3)$$

A low color variance score indicates high consensus. Since the images are 24-bit RGB color quantities, we compute per pixel, the variance score for each of the color channels separately and then average them together into a single matching score. This score is an initial estimate of the location's match likelihood. We also record the mean color $\bar{c}$ at each pixel location. The mean color is propagated through the other rendering stages to the synthesis stage. We indicate locations that are outside of the visual hull by assigning them the highest (worst) possible matching score and a default background color. The output of this stage is an image where the RGB channels contain the average color $\bar{c}$, and the alpha channel stores c_{var} as the initial matching scores.

Template Library

Since pixel color is not unique, establishing correspondences using only pixel color is not robust and may lead to many false matches. Matching locations where the coincident pixels are not in correspondence may be incorrectly assigned favorable matching scores based entirely on their color consensus. These false matches manifest as noise in the matching score image. The focus of this section is our technique for removing and reducing the incorrect matches.

We address this problem by examining the local pixel neighborhood around each location to find support for the match. We assume that the matching score function over the image varies continuously and that pixels in the local neighborhood should have similar matching scores (i.e. the local neighborhood of an incorrect match should contain many identifiably incorrect matches). We compute the mean matching scores of the neighboring pixels and use this average as the aggregated matching score for each location. We implement this step as an image convolution of the initial matching score image with a with a filter from the template library. This filter describes the pixel neighborhood and each of their respective pixel weights.

Choosing a good template filter is important for establishing correct matches. The traditional square template centered about the pixel position fails to find correct support when the pixel is located close to a surface discontinuity (Figure 4a). The square template assumes that the matching scores in this neighborhood vary smoothly, which is inconsistent with the surface discontinuity. Consequently, the neighborhood will contain bad pixel matches that are off of the object surface. They contribute high individual matching scores that will artificially increase the aggregated matching score. However, we may instead look for a different pixel neighborhood that does not include any discontinuities. A template with better neighborhood support is shown in Figure 4b. Some authors [Ols02a,Berg01a] have suggested using robust statistics to adapt the shape of the template in establishing local support for each pixel. Our rendering system instead applies a library of template window shapes to estimate the correct support. This method is more suitable for implementation on graphics hardware. The library consists of standard square template windows with different orientations.

Due to constraints on hardware resources, we use basic 7x7 square templates with the rotations

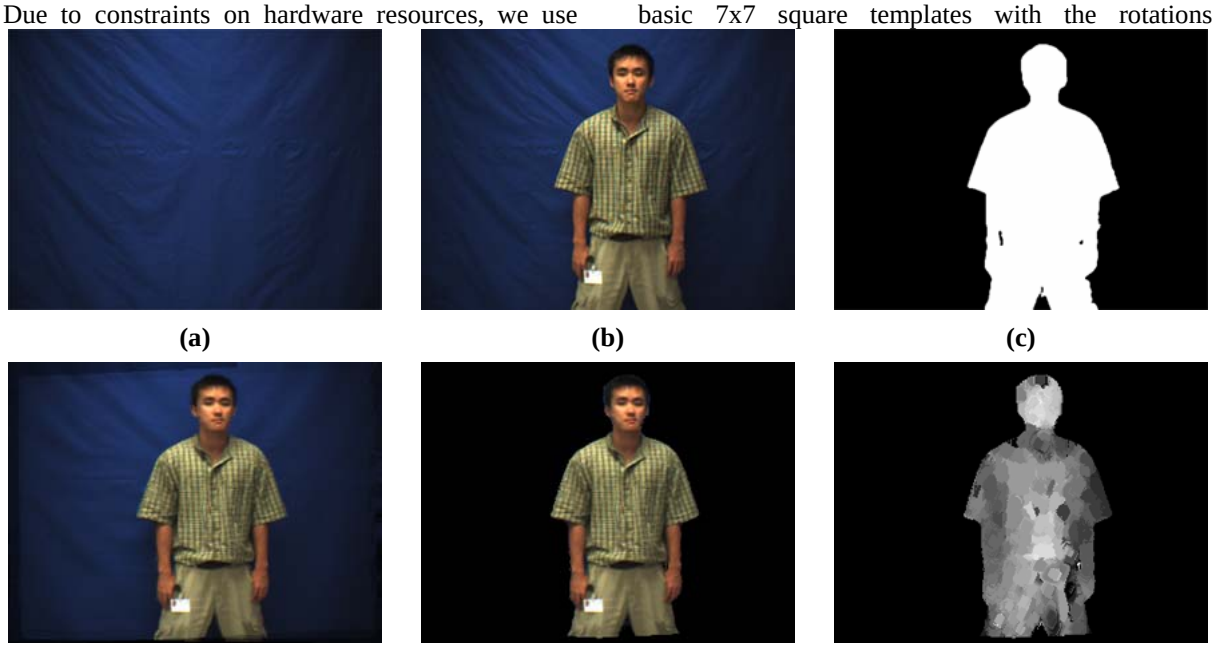


Figure 5. Results: (a) background image of input viewpoint (b) image from input viewpoint (c) segmented silhouette bit-mask (d) image from new viewpoint using multi-view stereo (e) image from new viewpoint rendered using multi-view stereo & visual hull (f) depth image

discretized to 0^o, 22.5^o, 45^o, and 67.5^o. Inspired by Kanade's sliding window technique [Oku91a], our library also includes copies of these four basic templates shifted with different pixel centers over the space of 7x7 integer shift vectors. Our library consists of 196 (7x7x4) unique shifts and orientations of the square template. The templates are applied to our initial matching score image and at each pixel we record the best matching score among the set of templates. To avoid wasteful computation, the templates in the library are only applied to pixels that are inside of the visual hull; all other pixels are discarded from further processing.

We apply each template in the library to the matching score image in two stages: First, we aggregate the matching scores over the four basic template windows. This separable convolution is implemented in two rendering passes. The matching score image is sampled bilinearly by the texture hardware for each template rotation. The aggregated scores for the four basic rotated templates are stored into the respective RGBA color channels. In the second stage, each template window is shifted to find a better matching score; we choose the best matching score among the shifted templates for each pixel. This step is implemented as a separable two-pass image convolution by a MIN filter on each color channel independently. We apply this filter over the neighborhoods of each of the four basic template shapes. Again, the graphics hardware samples the aggregated matching score image bilinearly over each rotated template window. Finally, the best matching score from the four template shapes for each pixel is chosen (minimum score among the RGBA channels)

Image Synthesis

Each plane associates an image that contains the mean color and aggregated matching score for each pixel. These images are rendered into the Z-Buffer as they are computed, using the matching score as depth. At each location, the Z-Buffer selects the pixel with the lowest matching score among the set of planes. We directly display each color pixel and update the Z-Buffer as each plane is rendered. To meet interactive frame rate requirements, we use a minimum set of planes for rendering and update the results by incrementally inserting more planes as additional processing time become available.

5. RESULTS

Our test scene consists of an actor against a blue curtain to simplify the foreground segmentation. We use 6 DragonFly FireWire cameras calibrated with in-house software that relies on feature detection and matching. The scene was sampled using four nearby cameras. Each camera records an initial image of the background and subtracts this against all subsequent image frames to automatically determine the actor's silhouette (Figure 5c). We use the input image

frames to render the scene from a new viewpoint (d). Note the ghosting and blur artifacts around the actor's profile. We can use the silhouettes to constrain the multi-view matching with the visual hull and effectively remove artifacts along the silhouette profile (Figure 5e). Our rendering system achieves a rendering rate of 15 fps at an image resolution of 320 x 240, discretizing the scene to 20 planes. The planes share the same orientations; the normals are perpendicular to the viewing direction, and the plane positions are distributed in equal steps in inverse depth as measured from the viewpoint.

We compare our template library against the standard 7x7 template, and Kanade's sliding window in Figure 6. We can see that the template library is more effective at recovering the depth of discontinuities along the silhouette profiles. Furthermore, by comparing the depth maps, we can see that the depth recovered by the square window is noisy along the profiles, and that the sliding window tends to produce "blocky" depth maps. The images produced by the template library filters shows significant improvement over the previous two techniques.

We give additional results in Figure 7. The input images were taken from a digital camera. The camera pose for each image was recovered using the in-house calibration software. The silhouette segmentation was computed by hand using Adobe Photoshop to remove the back walls.

6. CONCLUSIONS & FUTURE WORK

We have presented and demonstrated results on a rendering system that combines the strengths of visual hull rendering and multi-view stereo. We synthesize images that contain fewer artifacts than either approach. In our experiments, we rely on square templates to aggregate the matching score. We would like to investigate different template shapes to establish the neighborhood support. Furthermore, our current hardware implementation limits the size of the template window to 7x7. We plan to incorporate multi-resolution into our system to accommodate stereo matching with larger template window sizes. We would also like to expand the system to segment, track, and render multiple visual hulls.

7. ACKNOWLEDGMENTS

This work has been co-sponsored by the Advanced System Technology laboratory of STMicroelectronics and the Digital Media Innovation Program (DiMI) from the University of California.

8. REFERENCES

[Bak03a] Baker, S., Terence, S. and Kanade, T. 2003. "When Is the Shape of a Scene Unique Given Its Light-Field: A Fundamental Theorem of 3D Vision?" IEEE Transaction on Pattern Analysis and Machine Intelligence, 25, 2, 100-109.

[Berg01a] Berg A and Malik J, "Geometric Blur for Template Matching." CVPR 2001 601-614.

[Frag03a] ARB_fragment_program, http://oss.sgi.com/projects/ogl-sample/registry/ARB/fragment_program.txt

[Hart00a] Hartley R. and Zisserman, A. 2000. "Multiple View Geometry," Cambridge University Press.

[Kang01a] Kang, S.B., Szeliski, R. and Chai, J. 2001. "Handling occlusions in dense multi-view stereo," Proceedings IEEE Conference on Computer Vision and Pattern Recognition 2002, I:156-161.

[Laur94a] Laurentini, A. "The Visual Hull Concept for Silhouette Based image Understanding." IEEE PAMI 16.2 (1994), 150-162.

[Li03a] Li M, Magnor M, and Seidel H, "Hardware-Accelerated Visual Hull Reconstruction and Rendering." Graphics Interface'2003. (2003),

[Li03b] Li M, Magnor M, and Seidel H, "Improved Hardware-Accelerated Visual Hull Rendering." Vision, Modeling, and Visualization 2003.

[Mat01a] Matusik Wojciech, Buehler C, and McMillan L, "Polyhedral Visual Hulls for Real-Time Rendering," 12th Eurographics Workshop on Rendering (2001), 115-125.

[Mat00a] Matusik Wojciech, Buehler C, Raskar R, McMillan L, and Gortler S, "Image-Based Visual Hulls." SIGGRAPH 2000.

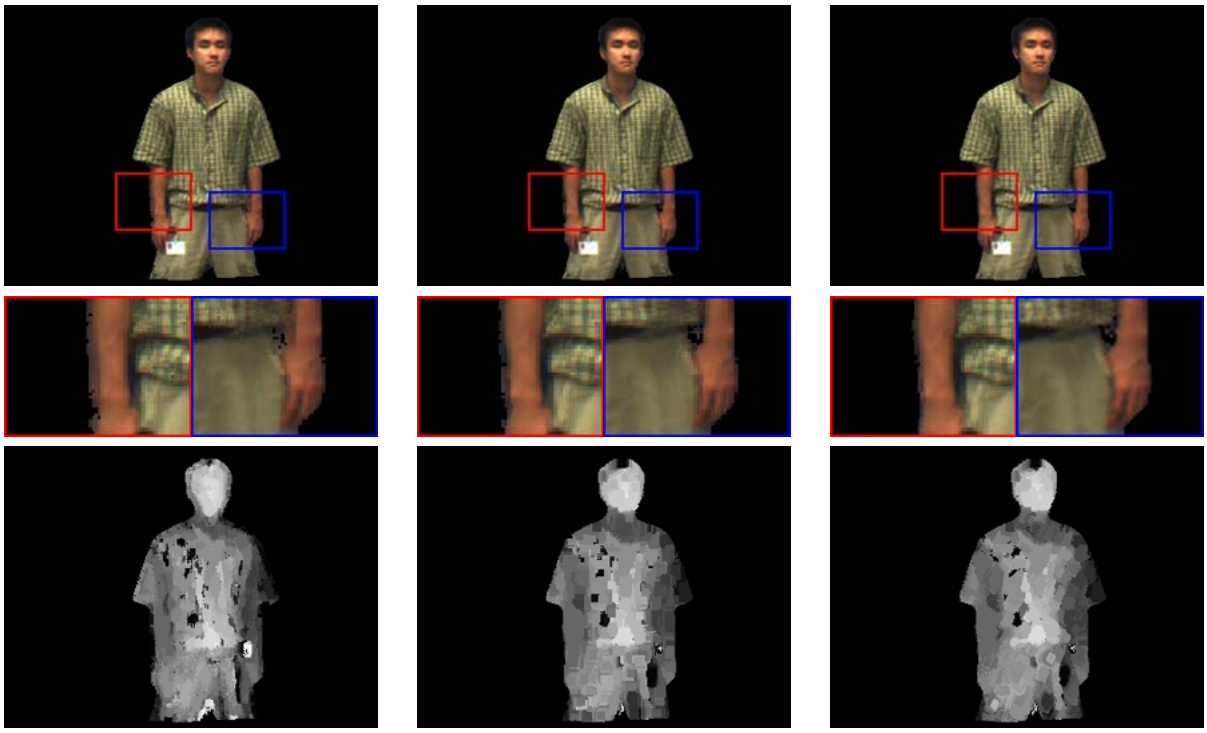
[Okut91a] Okutomi M and Kanade T, "A multi-baseline stereo." CVPR 1991.

[Ols02a] Olson C, "Maximum-Likelihood Image Matching." IEEE PAMI 24.6 (2002), 853-857.

[Prin02a] Prince, S.J.D., Xu, K. and Cheok, A.D. 2002. "Augmented Reality Camera Tracking with Homographies," Computer Graphics and Applications, 22, 6, 39-45.

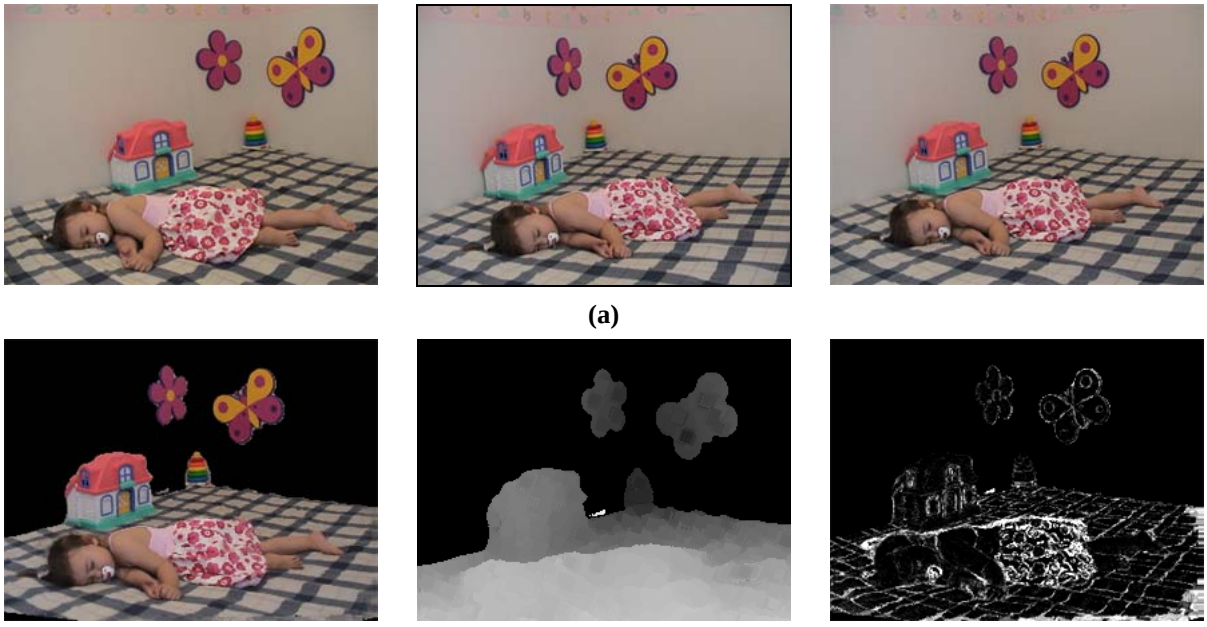
[Yan02a] Yang, J.C., Matthew, E., Buehler, C. and McMillan, L. 2002. "A Real-Time Distributed Light Field Camera," Proceedings Eurographics Workshop on Rendering 2002, 77-85.

[Yan00a] Yang R, Welch G, and Bishop G, "Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware." PacificGraphics 2000.



(a) **(b)** **(c)**

Figure 6. Comparison of filters: (a) 7x7 square template (b) sliding window (c) template library



(a)

(b) **(c)** **(d)**

Figure 7: Baby dataset rendered using 20 sampling planes: (a) 3 images from input sequence (b) synthesized image at novel viewpoint (c) depth map (d) final composited matching score image