Image-Based Photo Hulls

Gregory G. Slabaugh, Ronald W. Schafer, Mat C. Hans
Client and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-28
February 4th, 2002*

3D photography, photo hull, visual hull, image-based rendering, new view synthesis, photo-consistency, real-time computer vision

We present an efficient image-based rendering algorithm that computes photo hulls of a scene photographed from multiple viewpoints. Our algorithm, *Image-Based Photo Hulls* (IBPH), like the *Image-Based Visual Hulls* (IBVH) algorithm from Matusik et. al. on which it is based takes advantage of epipolar geometry to efficiently reconstruct the geometry and visibility of a scene. Our IBPH algorithm differs from IBVH in that it utilizes the color information of the images to identify scene geometry. These additional color constraints result in more accurately reconstructed geometry, which projects to better synthesized virtual views of the scene. We demonstrate our algorithm running in a realtime 3D telepresence application using video data acquired from four viewpoints.

Image-Based Photo Hulls

Gregory G. Slabaugh, Ronald W. Schafer, Mat C. Hans

January 25, 2002

Abstract

We present an efficient image-based rendering algorithm that computes photo hulls of a scene photographed from multiple viewpoints. Our algorithm, *Image-Based Photo Hulls* (IBPH), like the *Image-Based Visual Hulls* (IBVH) algorithm from Matusik et. al. on which it is based, takes advantage of epipolar geometry to efficiently reconstruct the geometry and visibility of a scene. Our IBPH algorithm differs from IBVH in that it utilizes the color information of the images to identify scene geometry. These additional color constraints result in more accurately reconstructed geometry, which projects to better synthesized virtual views of the scene. We demonstrate our algorithm running in a realtime 3D telepresence application using video data acquired from four viewpoints.

Keywords: 3D photography, photo hull, visual hull, image-based rendering, new view synthesis, photo-consistency, real-time computer vision.

1 Introduction

The task of generating a photo-realistic 3D representation of a visual scene is an important and challenging problem. Debevec et. al. [1] demonstrated in their Campanile movie that it is possible, using a user-assisted 3D modelling program and a handful of photos of a college campus, to produce a digital model of the scene that when rendered, yields images of stunning photorealism from novel viewpoints. Since this work, there has been much interest in producing results of similar quality using algorithms that are automatic and work on scenes composed of surfaces of arbitrary geometry.

Recently, researchers have become interested in reconstructing time-varying scenes [2, 3, 4, 5, 6]. Unfortunately, most standard approaches to the 3D scene reconstruction problem such as multi-baseline stereo, structure from motion, and shape from shading are too slow for realtime application on current computer hardware. When working with multi-view video data, most techniques perform the 3D reconstruction offline after the images have been acquired. Once the reconstruction is complete, it is rendered in realtime.

A notable exception is the Image-Based Visual Hulls (IBVH) algorithm [7], developed at MIT by Matusik et. al. This algorithm is efficient enough to reconstruct and render views of the scene in realtime. The key to this algorithm's efficiency is the use of epipolar geometry for computing the geometry and visibility of the scene. By taking advantage of

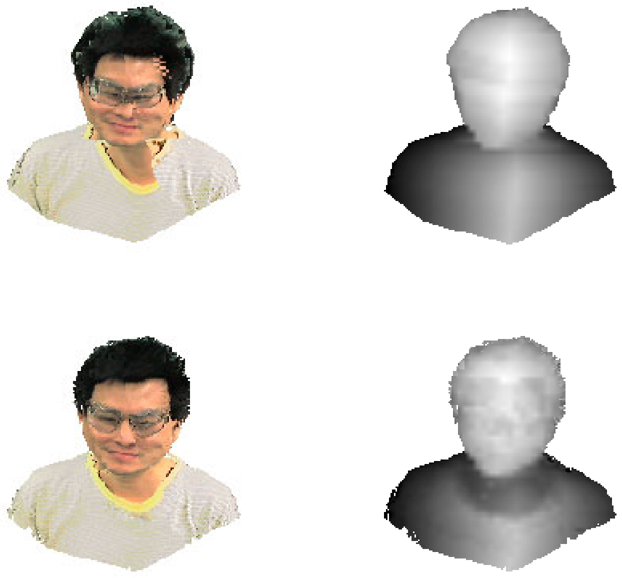

Figure 1: Visual hull reconstruction (upper row) vs. photo hull reconstruction (lower row). Left to right: synthesized view and depth map.

epipolar relationships, all of the steps of the algorithm function in the image space of the photographs (also called *reference views*) taken of the scene.

While the IBVH algorithm is exceptionally efficient, the geometry it reconstructs is not very accurate. This is because the IBVH algorithm only reconstructs the visual hull of the scene. The visual hull is a conservative volume that contains the scene surfaces being reconstructed. When photographed by only a few cameras, the scene's visual hull is much larger than the true scene. Even if photographed by an infinite number of cameras, many scenes with concavities will not be modelled correctly by a visual hull. One can partially compensate for such geometric inaccuracies by view-dependent texture-mapping (VDTM), as done in the IBVH approach.

However, artifacts resulting from the inaccurate geometry are still apparent in new synthesized views of the scene, as shown in Figure 1. This figure demonstrates a reconstruction of a person's head photographed from four viewpoints. A new view of the scene, placed halfway between two reference views, is rendered from the reconstruction. The top row shows the visual hull reconstruction. At this viewpoint, the right side of the face is texture-mapped with one reference image, while the left side of the face is texture-mapped with another. Due to the geometric inaccuracy of the visual hull, there is a salient seam along the face where there is a transition between the two images used to texture-map the surface. The improved geometry of the photo hull corrects this problem, as shown in the bottom row of the figure.

In this paper we adapt the IBVH algorithm to reconstruct photo hulls. The photo hull is also a volume that contains the scene surfaces being reconstructed. However, it is a tighter fit to the true scene geometry than the visual hull. That is,

$$\text{True Scene} \subseteq \text{Photo Hull} \subseteq \text{Visual Hull}$$

New views synthesized using the more accurate geometry of the photo hull have improved photorealism. Like IBVH, all the steps of our algorithm function in image space (hence we call our algorithm *Image-Based* Photo Hulls). Our approach combines the efficiency of the IBVH algorithm with the improved geometric accuracy of the photo hull.

2 Related Work

2.1 Visual Hulls

A standard approach to reconstructing a 3D object using multi-view images is to compute the visual hull [8]. For each reference view, a silhouette is generated by segmenting the photograph into foreground and background. Foreground pixels correspond to points to which the 3D object projects. Everything else is background.

Each silhouette constrains the 3D space in which the object is located. If a 3D point projects to background in any of the images, it cannot be part of the 3D object being reconstructed. After eliminating such points, the surface of the region of space that remains is the *visual hull*. The visual hull is guaranteed to contain the 3D object. Using more reference views produces a visual hull that more closely resembles the geometric shape of the true 3D object. However, even with an infinite number of photographs, the visual hull cannot model surface concavities that are not apparent in the silhouettes.

A variety of algorithms have been developed to compute the visual hull of a scene. Perhaps the most common approach is to operate in a volumetric framework. A good example is given in [9]. A volume that contains the scene being reconstructed is defined. The volume is then tessellated into voxels. All the voxels that project to solely background pixels in one or more reference views are removed (carved). The remaining voxels represent the visual hull and its interior. Such an algorithm can adopt a multi-resolution strategy to achieve faster results.

Recently, the Image-Based Visual Hulls [7] algorithm was developed and demonstrated to produce realtime new views of a scene. Our IBPH algorithm is based on the IBVH algorithm, so we will briefly review IBVH in Section 3.

2.2 Photo Hulls

The problem of reconstructing a 3D model of a scene using multiple 2D photographs is ill-posed. For a given set of 2D photographs, multiple 3D models that reproduce the photographs can and often do exist. Kutulakos and Seitz [10] introduce the *photo hull*, which is the largest shape that contains all reconstructions in the equivalence class of 3D models that reproduce the photographs. The photo hull is unique and is relatively simple to compute. New, photo-realistic views of the scene can be synthesized by rendering the photo hull to virtual viewpoints.

A number of algorithms that compute photo hulls have been developed [11, 10, 12, 13]. These methods utilize photo-consistency [10] as a constraint to identify scene surfaces. A point in space is said to be *photo-consistent* if (1) it does not project to background and (2) when visible, the light exiting the point (i.e. radiance) in the direction of each reference view is equal to the observed color in the photograph.

For simplicity, one often assumes that the scene is Lambertian, although this isn't strictly necessary. Under this assumption, a point on a scene surface will project to a similar color in each reference view. The photo hull is then computed by finding the spatially largest set of points in 3D space that project to matching colors in the reference images.

These algorithms begin with a voxel space of initially opaque voxels that encompasses the scene to be reconstructed. As the algorithms run, opaque voxels are tested for photo-consistency and those that are found to be inconsistent are carved, i.e. made transparent. Convergence occurs when all the remaining opaque voxels are photo-consistent. When these final voxels are assigned the colors they project to in the input images, they form a model that closely resembles the scene. This model is the photo hull.

Our IBPH algorithm adopts a similar strategy to compute views of the photo hull. The algorithm starts with a surface larger than the scene, and then iteratively carves space using a photo-consistency measure until the visible reconstructed points become photo-consistent. Our approach differs from previous photo hull reconstruction algorithms in that the algorithm functions in image space and produces a view-dependent reconstruction. Unlike [11, 10, 12, 13, 14], our approach does not employ a static 3D voxel space for reconstruction. Rather, the reconstructed geometry is computed in a space defined by a virtual camera, and changes as the camera is moved about the scene. Rather than reconstruct the full photo hull geometry, our method only reconstructs the portion of the photo hull that is visible to the virtual camera.

For an overview of methods that reconstruct visual and photo hulls, please refer to [15, 16].

Notation

Before proceeding to the explanation of our algorithm, let us define some notation. A point in 3D space is represented in homogeneous coordinates by a boldface capital letter, such as $\mathbf{P}$. In this paper, $\mathbf{P} = [\ x \quad y \quad z \quad w\ ]^T$. The projection of this point into an image is a 2D point represented in homogeneous coordinates by a boldface lowercase letter, such as $\mathbf{p} = [\ x \quad y \quad w\ ]^T$. To convert a homogeneous image point to inhomogeneous coordinates (i.e. pixel coordinates), one simply divides $\mathbf{p}$ by the w component. Thus, a pixel will have coordinates $\mathbf{p} = [\ x/w \quad y/w \quad 1\ ]^T$.

3 Image-Based Visual Hulls

In this section, we briefly review the IBVH algorithm. In the following section, we will show how we extend this algorithm to reconstruct photo hulls.

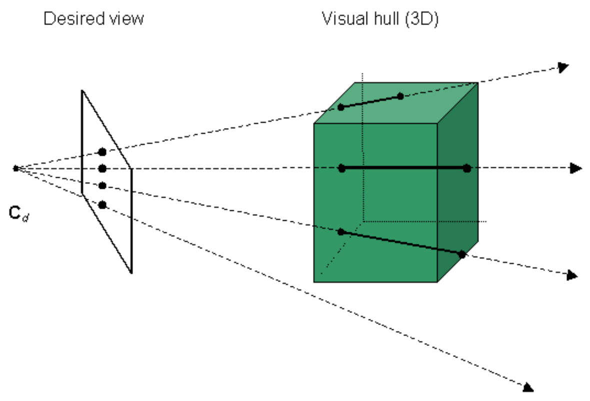

Figure 2: View-dependent geometry.

3.1 Computing geometry

One of the unique properties of the IBVH algorithm is that the geometry it reconstructs is view-dependent. A user moves a virtual camera about the scene. For each virtual camera placement (also called *desired view*), the IBVH algorithm computes the extent that back-projected rays from the center of projection $\mathbf{C}_d$ intersect the visual hull in 3D space. This is shown in Figure 2. Thus, the representation of the geometry is specified for the desired view, and changes as the user moves the virtual camera.

Consider an individual ray, as shown in Figure 3. The ray is back-projected from the desired view's center of projection, through a pixel in the image plane, and into 3D space. This ray projects to an epipolar line in each reference view. The IBVH algorithm determines the 2D intervals where the epipolar line crosses the silhouette. These 2D intervals are then "lifted" back onto the 3D ray using a simple projective transformation. The intervals along the 3D ray from all reference views are intersected. The resultant set of intervals describe where the ray pierces the visual hull. These are called *visual hull intervals* in this paper. In Figure 3, one visual hull interval is found along the back-projected ray. Once this procedure has been performed on all rays back-projected from the desired view, the reconstruction of the view-dependent geometry of the visual hull is complete.

3.2 Computing visibility

In order to color a point on the visual hull, it is necessary to determine which cameras have an unoccluded view of the point. Thus, visibility must be computed before texture-mapping the reconstructed geometry.

At a pixel $\mathbf{p}$ in the desired view, the first point (if any) along the first visual hull interval indicates a point $\mathbf{P}$ in 3D space that projects to $\mathbf{p}$ and is visible in the desired view, as shown in Figure 5 (a). To compute visibility, for each reference view we need to determine if $\mathbf{P}$ is visible. $\mathbf{P}$ must be visible in the reference view if the line segment $\overline{\mathbf{PC}_r}$ between $\mathbf{P}$ and the reference view's center of projection $\mathbf{C}_r$ does not intersect any visual hull geometry.

The layered depth image representation of the visual hull makes this easy to determine. In the desired view, $\overline{\mathbf{PC}_r}$ projects to an epipolar line segment $\overline{\mathbf{pe}}$, where $\mathbf{e}$ is the epipole,

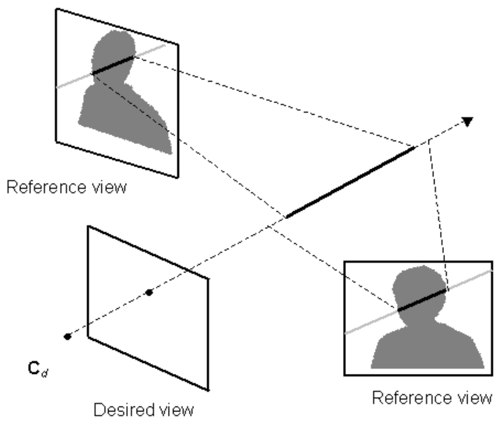

Figure 3: Determining a ray's visual hull intervals.

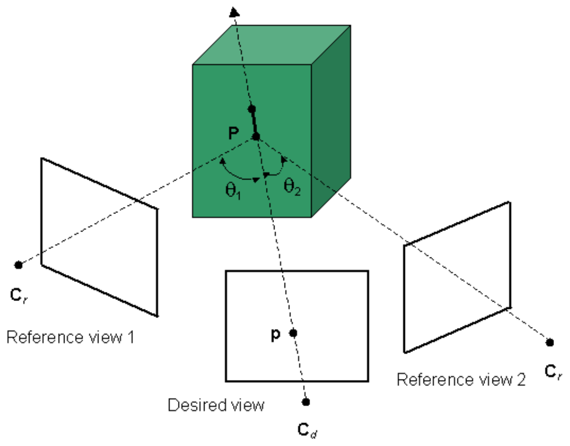

Figure 4: View-dependent texture-mapping.

found by projecting $\mathbf{C}_r$ into the desired view, as shown in Figure 5 (b). For each pixel along $\overline{\mathbf{pe}}$, the visual hull intervals can be checked to see if they contain geometry that intersects $\overline{\mathbf{PC}_r}$. If an intersection occurs, point $\mathbf{P}$ is not visible in the reference view, and no more pixels along $\overline{\mathbf{pe}}$ need be evaluated. Otherwise, one continues evaluating pixels along $\overline{\mathbf{pe}}$, until there are no more pixels to evaluate. If no visual hull interval has intersected $\overline{\mathbf{PC}_r}$, then the point $\mathbf{P}$ is visible in the reference view.

The IBVH paper [7] discusses discretization issues in computing visibility using this approach, as well as occlusion-compatible orderings to improve its efficiency.

3.3 View-dependent texture mapping

Once visibility has been computed, one can color the visual hull using the reference views. The IBVH paper employs view-dependent texture mapping, which retains view-dependent effects present in the photos, and works well with the inaccurate geometry of the visual hull.

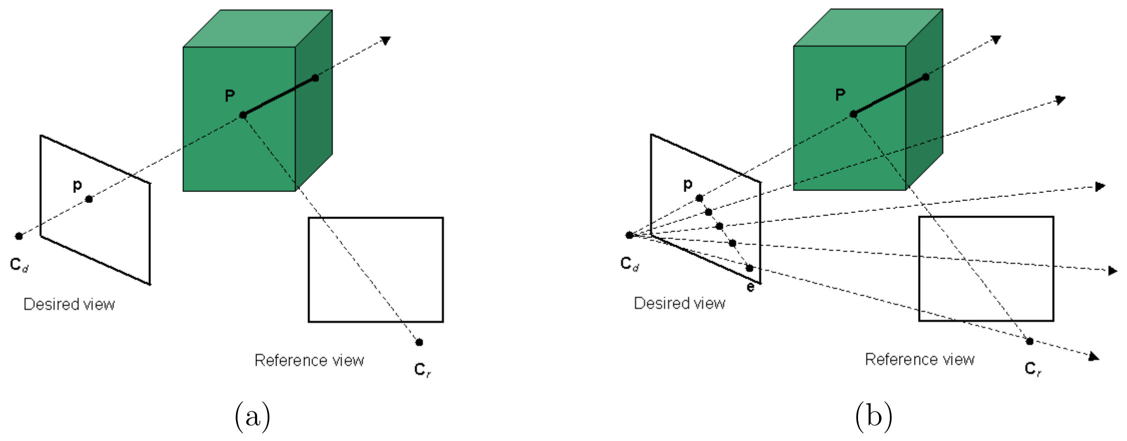

Figure 5: **P** is visible in the reference view if there is no occluding geometry along $\overline{\mathbf{PC}_r}$.

To color a point **p** in the desired view, the closest point **P** on the hull is found. Then, for each reference view that has visibility of **P**, the angle between $\overline{\mathbf{PC}_d}$ and $\overline{\mathbf{PC}_r}$ is found, as shown in Figure 4. The reference view with the smallest angle is chosen to color the visual hull. This is the reference view that has the "best" view of **P** for the virtual camera's location. For example, in Figure 4, reference view 2 would be chosen since $\theta_1 > \theta_2$.

4 Image-Based Photo Hulls

In this section we describe our new method of reconstructing photo hulls.

4.1 Approach

Our IBPH approach first computes the visual hull using the IBVH algorithm, which quickly eliminates a large portion of 3D space that does not contain scene surfaces. Our algorithm then evaluates the photo-consistency of the closest point on the visual hull along each ray back-projected from the desired view. If the point is inconsistent, we take a small step along the ray, moving away from the desired view, as depicted in Figure 6. We continue stepping along an inconsistent ray until it either becomes consistent or we have stepped beyond all visual hull intervals along the ray. This latter case indicates that there is no photo-consistent geometry along the ray.

Note that in this approach, only the points on the hull that are visible in the desired view are processed. These points are the first points in the first visual hull interval along each back-projected ray. The points farther along the ray are not visible and are not processed by our IBPH algorithm.

4.2 Photo-Consistency

To determine the photo-consistency of a 3D point **P** along a ray, we project **P** into the ith reference view, yielding an image-space point $\mathbf{p}_i$. We only perform this projection for the

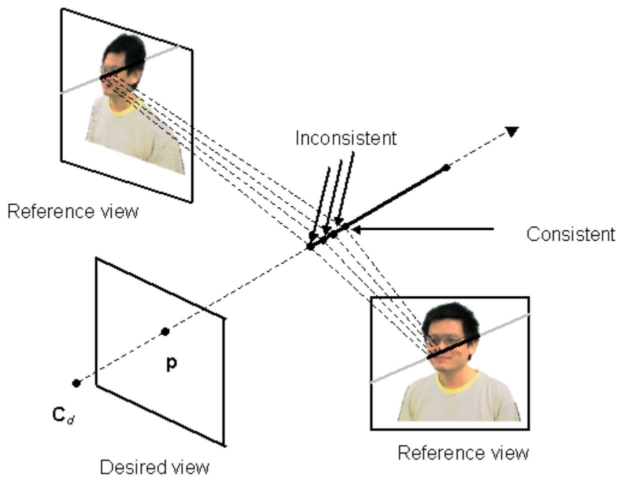

Figure 6: Computing the image-based photo hull.

reference views that have visibility of $\mathbf{P}$. Around each $\mathbf{p}_i$ we collect a small neighborhood of pixels, N_i to use in our color matching function.

There are many methods one can employ for matching color distributions to determine photo-consistency. A standard approach is to threshold the standard deviation of all the pixels. That is,

$$\text{consistency} = \begin{cases} \text{True, if } \sigma \leq T_1 \\ \text{False, otherwise} \end{cases} \tag{1}$$

where σ is the standard deviation of all pixels, $\sum_i N_i$, and T_1 is a user-defined threshold. In our work, we reconstruct the scene using RGB images, so a standard deviation is computed for each color channel and summed to produce σ.

This measure of photo-consistency works reasonably well for many scenes. However, it can perform poorly for surfaces that project to consistent, yet highly varying colors in the reference images. This can occur on edges as well as textured surfaces. In such a case, each reference image will observe multiple different colors for the surface. Consequently, the standard deviation will be high, the photo-consistency measure can incorrectly return false.

Textured surfaces and edges will project to pixels with a high standard deviation in *each* image. We use this fact to modify the consistency measure described above to handle such surfaces. Let the standard deviation of a set of pixels N_i from a reference view be σ_i. Our new photo-consistency measure is then

$$\text{consistency} = \begin{cases} \text{True, if } \sigma \leq T_1 + \overline{\sigma} T_2 \\ \text{False, otherwise} \end{cases} \tag{2}$$

where $\overline{\sigma}$ is the average of σ_i and T_2 is a second user-defined threshold.

This consistency measure simply adds an additional term $\overline{\sigma} T_2$ to the one defined in Equation 1. This term spatially adapts the consistency measure based on the colors observed in the 3D point's projection. The value of $\overline{\sigma}$ will be small when the 3D point projects to homogenous colors in each image. In this case, there will be little difference between the two consistency measures 1 and 2. If these colors are similar, the point will be declared

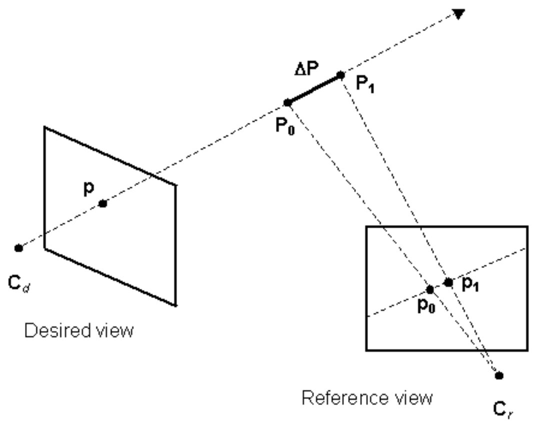

Figure 7: Stepping along an epipolar line.

consistent. If these colors are dissimilar, the point will be declared inconsistent. When the point projects to highly varying pixels in each image, the $\overline{\sigma}$ term will increase the maximum value of σ allowable for the point to be declared consistent. This allows for textured surfaces, as well as edges, to be correctly reconstructed. It also eases the Lambertian assumption. The threshold T_2 allows one to weight the contribution of this term to the photo-consistency measure.

This two-parameter consistency measure is the one used to produce photo hulls in this paper.

4.3 Stepping Along Epipolar Lines

As we step in 3D along an inconsistent ray, looking for the point at which it becomes consistent, we must simultaneously step along an epipolar line in each reference view. The brute-force way of stepping along the epipolar line in a reference view is to simply project each 3D point $\mathbf{P}_i$ on the ray to the reference view point $\mathbf{p}_i$ by multiplying the reference view's projection matrix H with $\mathbf{P}_i$, i.e. $\mathbf{p}_i = H\mathbf{P}_i$. Such an approach will work, but will require a large number of matrix multiplications.

While the step size $|\Delta\mathbf{P}|$ in 3D is constant, the step size between adjacent points along the epipolar line in a 2D reference view varies due to the projection. However, since the projection is a homography (linear projective transformation), the step size *is* constant in homogeneous coordinates. We use this fact to produce a more efficient procedure for stepping along the epipolar line.

Consider the 3D point $\mathbf{P}_0$ on the ray, as shown in Figure 7. It projects to a point $\mathbf{p}_0 = H\mathbf{P}_0$ in a reference image. If we take a step along the ray, we arrive at a 3D point $\mathbf{P}_1 = \mathbf{P}_0 + \Delta\mathbf{P}$. The point $\mathbf{p}_1$, the projection of $\mathbf{P}_1$ into the reference view can be written as

$$\begin{aligned} \mathbf{p}_1 &= H\mathbf{P}_1 \\ &= H(\mathbf{P}_0 + \Delta\mathbf{P}) \\ &= \mathbf{p}_0 + H\Delta\mathbf{P} \end{aligned}$$

Thus, we can incrementally update the homogeneous position of the point along the epipolar line in the reference view. That is,

$$\mathbf{p}_i = \begin{cases} H\mathbf{P}_0, & i = 0 \\ \mathbf{p}_{i-1} + H\Delta\mathbf{P}, & i > 0 \end{cases} \tag{3}$$

We pre-compute $H\Delta\mathbf{P}$ for each ray and store it in a look-up table. As we step along the epipolar line, we use Equation 3 to compute the homogeneous position of the point $\mathbf{p}_i$. With this approach, stepping along an epipolar line is very efficient.

4.4 IBPH Visibility

When evaluating the photo-consistency of a 3D point, only pixels from the reference views that have visibility of the 3D point should be used. As one steps along the inconsistent rays, the visibility of the scene may change. A point that was not visible in a reference view before may become visible after the step is taken. Therefore, it is necessary to update visibility after stepping. This is achieved by re-executing the visibility procedure described in Section 3.2.

Visibility could be updated each time a step is taken along each ray. However, such an excessive number of visibility updates results in a slow reconstruction. Instead, our algorithm takes one step along each inconsistent ray, and then updates visibility. As a result, the visibility may be out-of-date when evaluating some 3D points. However, such an approach is conservative. Pixels from only a subset of the references views that have visibility of the point will contribute to the consistency measure. This may result in some 3D points being erroneously classified as consistent, while a full visibility calculation would show that they are really inconsistent. Such an approach is similar to that used in the GVC-IB [12] algorithm.

One option for trading off speed versus accuracy in our approach is to not update visibility at all. When computing photo-consistency, one can use the visibility of the visual hull, which can be computed once after executing the IBVH algorithm. Thus, visibility becomes out-of-date once a step has been taken along a ray. Using this inexact visibility will not yield the best reconstruction possible, but will be fast. In practice we have found that such an approach results in a slight degradation in quality for the data sets that we have tested, but runs approximately 25% faster.

4.5 Sampling

To improve reconstruction efficiency, both the IBVH and IBPH algorithms can be applied in a multi-resolution fashion[1]. The algorithm is executed not for every pixel in the desired view, but rather on a coarse raster. One first computes the hull at sampling locations $(x \cdot DX, y \cdot DY)$ in the desired image, where DX and DY are constants. These sampling locations are shown as black dots in Figure 8. For in-between pixels on the boundary, indicated using black squares in the figure, the hull is computed at higher resolution. For pixels inside the boundary, the closest point of the visual hull interval is interpolated from

[1]http://graphics.lcs.mit.edu/∼wojciech/vh/ctof.html

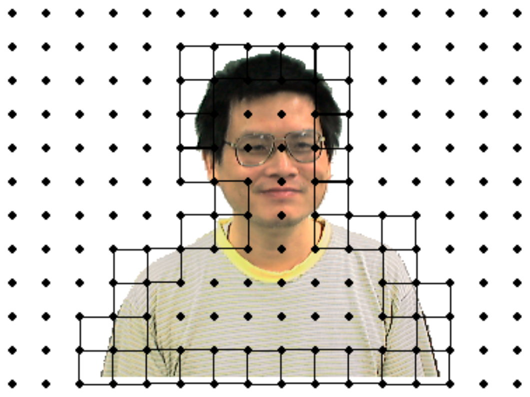

Figure 8: Multi-resolution hull reconstruction.

adjacent samples. This approach significantly reduces the number of rays that must be processed, resulting in a faster reconstruction. In our application, typically DX and DY are both set to 4.

For IBPH, there is an additional sampling parameter, $\Delta\mathbf{P}$. This is the size of the 3D step taken along an inconsistent ray. In our application, we set $|\Delta\mathbf{P}|$ to a size of 2.5 mm, which for our camera configuration, results in a projected size $|\Delta\mathbf{p}|$ of about 1 pixel for most reference views.

4.6 Convergence

The IBPH algorithm steps along the inconsistent rays, stopping at the point at which each ray becomes photo-consistent. For convergence, one can require that all rays are photo-consistent. However, often during a reconstruction, a significant majority of the rays will become consistent quickly. Continuing to process a handful of inconsistent rays will yield little impact on the overall quality of the reconstruction, but can take a lot of time. In our application, we have introduced a mechanism to terminate the reconstruction when M or less rays are inconsistent. When M is a small number, good quality hulls are produced quickly.

Figure 9 justifies our use of M to terminate the reconstruction before all rays are consistent. This plot shows ray classifications versus iteration for a synthetic view of our Sam data set. The visual hull projected to 1333 of the 80 x 60 points on the coarse raster. Rays back-projected through these points were analyzed using the IBPH algorithm. Initially, 635 were inconsistent and 698 were consistent, as shown in the figure. At each iteration of the algorithm, a step was taken along each inconsistent ray. The plot of the number of inconsistent rays is very steep at first, indicating that many rays become consistent quickly. After 60 iterations, most rays are consistent. However, it takes an additional 140 iterations for the few remaining inconsistent rays to become consistent. For realtime application, one would rather not continue processing these rays, as they will not significantly contribute to the quality of the reconstructed model.

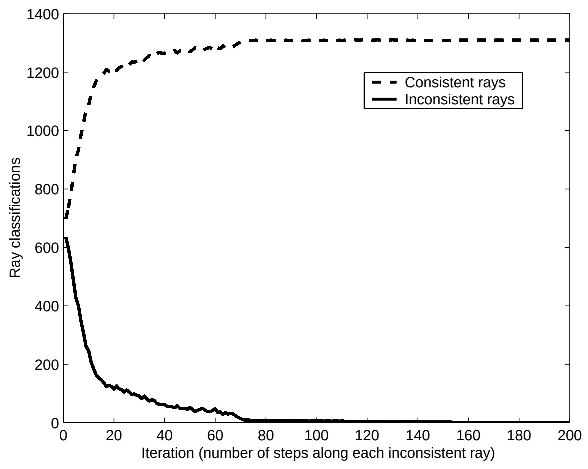

Figure 9: Ray classifications vs. iteration.

4.7 Spatial Coherence

Most scenes exhibit a high degree of spatial coherence, as they consist of surfaces that do not radically change their position over a small region of space. Accordingly, many stereo vision algorithms impose a regularization criterion that requires the reconstructed geometry to be smooth. In a similar vein, we have developed a very simple and efficient smoothing technique that we incorporate into our IBPH algorithm. The smoothing also helps mitigate reconstruction errors due to noise and specularities.

When stepping along an inconsistent ray, we keep track of the number of steps we have taken, k. Before taking another step, we compare k to a local mean computed by averaging the number of steps taken along rays in a small neighborhood around the inconsistent ray. We denote this local average $\bar{k}$. If $k > \bar{k} + K$, where K is a small constant, we do not step along the ray. This ensures that the number of steps taken along a ray is not significantly different from that of its neighbors, resulting in a surface that is spatially coherent. This smoothing approach requires very little computation and works naturally with the representation of the hull geometry used in our algorithm.

Figure 10 shows the effect of changing K for a reconstruction of our Sam data set. Notice that there are less abrupt transitions in the depth map in the top-most reconstruction ($K = 1$) compared to the bottom-most reconstruction ($K = 5$).

4.8 Homogeneous Surfaces

Surfaces that have a homogeneous color are difficult to properly reconstruct using color matching methods. A point in space near the surface will project to similar colors in the reference images. Such a point will be photo-consistent even though it is not actually on the surface being reconstructed. Our algorithm is not immune to this problem, as is visible in any of the reconstructions of our Sam data set, such as those in Figure 10. The depth map indicates some extra geometry jutting out of the person's chest, resulting from a homogeneously colored shirt the person was wearing. However, geometrical inaccuracies due to

Figure 10: Varying K. From top to bottom, $K = 1$, 2, and 5.

```
compute IBVH
compute visibility
pre-compute homogeneous ray steps $H\Delta\mathbf{P}$ in each reference image
do
    evaluate photo-consistency
    for each inconsistent ray in desired view
        if (number of steps along ray $k <= \bar{k} + K$)
            step along inconsistent ray
        else
            set ray consistent
    if (updating visibility)
        update visibility
} while(number of inconsistent rays > $M$)
display hull using VDTM
```

Figure 11: Pseudo-code for the IBPH algorithm. See text for details.

homogeneous surfaces are not significant once the model is texture-mapped. For example, the seams shown in Figure 1 will not be present because the geometry will project to a homogeneous color in each reference view.

4.9 Pseudo-Code

The pseudo-code for our IBPH algorithm appears in Figure 11.

5 Results

We have implemented the IBPH algorithm on a multi-camera system, similar to the one developed at MIT for the IBVH algorithm. We have four calibrated Sony DFW-V500 digital cameras. The cameras are synchronized so that they take images of the scene at the same instant of time. Each camera is connected to an 800 MHz HP Kayak XM600 machine running Windows 2000. These machines perform background subtraction on the incoming frames, segmenting them into foreground and background regions. The resolution of the reference images is 320 x 240 pixels.

The segmented reference images are sent over a 100 Mb/s switch to our server machine, which computes the 3D reconstruction. Our server machine is a dual processor 2 GHz HP x4000 workstation running Windows 2000. Our algorithm has been multi-threaded to take advantage of our multi-processor machine. The ith thread reconstructs the scene using a set of images corresponding to time t_i. In this way, the IBPH algorithm can very naturally be parallelized.

Figure 12 shows a view from a realtime 3D telepresence application we are currently developing. The 3D model of the person's head and upper body is reconstructed online

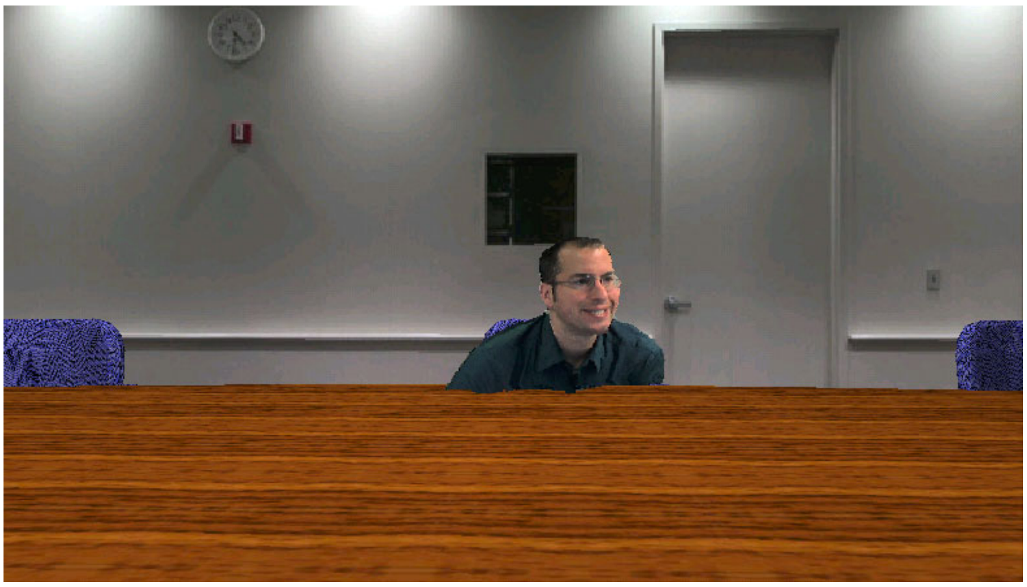
Figure 12: Using IBPH in a realtime 3D telepresence application.

using the IBPH algorithm, but using pre-recorded data. For this example, the parameters used were $DX = 4$, $DY = 4$, $M = 60$, $K = 2$, and $|\Delta\mathbf{P}| = 2.5$ mm. The model of the person is then depth-composited with a 3D model of a conference room. New synthesized views of this composited scene are generated at 7.5 frames per second.

6 Conclusion

In this paper we have presented our Image-Based Photo Hulls algorithm that efficiently reconstructs photo hulls. Our algorithm extends IBVH by utilizing the color information in the reference views to further constrain the 3D space containing scene surfaces. The more accurate geometry reconstructed by our technique results in better new views synthesized of the scene.

There are some limitations to our approach. While the photo-consistency measure described in Section 4.2 has some tolerance for non-Lambertian scenes, it fails for scenes that have significant specularities. One would have to adopt a different photo-consistency measure to properly reconstruct specular surfaces. For fairness, we should note that visual hull reconstruction algorithms like IBVH do not have a problem with non-Lambertian scenes, as they do not perform color matching. For the data sets shown in this paper, the IBPH algorithm requires approximately five times the computation of the IBVH algorithm. This extra computation is necessary since the IBPH algorithm must analyze each ray to determine at what point it becomes photo-consistent.

7 Future work

There are several future directions we are interested in pursuing. First, while our telepresence application processes the images online, it currently only works using pre-recorded data. We are working to get the system to reconstruct live data taken with the multi-camera system.

The photo-consistency measure presented here does not take into account surface orientation or the distance of the reference view from the surface. Correlation-based matching similar to [17] might improve reconstruction quality. Additionally, we do not take into consideration temporal coherence. Motion constraints could be imposed to improve efficiency and reconstruction quality. Finally, an adaptive approach to determining $|\Delta\mathbf{P}|$, the step size taken along each ray might improve the efficiency of our algorithm. When the photo-consistency measure is *very* inconsistent, one might take larger steps. As the consistency improves, one could take smaller steps until the ray becomes photo-consistent.

8 Acknowledgements

We thank Wojceich Matusik, Chris Buehler, and Leonard McMillan of MIT for sharing the IBVH source code and system configuration with us. We also thank Bruce Culbertson, Tom Malzbender, and Irwin Sobel of HP Labs for several useful discussions regarding the IBPH algorithm. We thank Dan Gelb for producing the virtual conference room model used in Figure 12. We also acknowledge Fred Kitson for his support and encouragement.

References

[1] P. Debevec, C. Taylor, and J. Malik, “Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach,” in *SIGGRAPH*, 1996, pp. 11–20.

[2] Andrew C. Prock and Charles R. Dyer, “Towards real-time voxel coloring,” in *Image Understanding Workshop*, 1998, pp. 315–321.

[3] P. Narayanan, P. Rander, and T. Kanade, “Constructing virtual worlds using dense stereo,” in *ICCV*, 1998, pp. 3–10.

[4] S. Moezzi, L. C. Tai, and P. Gerard, “Virtual view generation for 3d digital video,” *IEEE Multimedia*, vol. 4, no. 1, pp. 18–26, 1997.

[5] S. Vedula, S. Baker, S. Seitz, and T. Kanade, “Shape and motion carving in 6d,” in *CVPR*, 2000, vol. 2, pp. 592–598.

[6] R. Carceroni and K. Kutulakos, “Multi-view scene capture by surfel sampling: From video streams to non-rigid motion, shape, and reflectance,” in *ICCV*, 2001, vol. 2, pp. 60–67.

[7] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan, “Image-based visual hulls,” in *SIGGRAPH*, 2000, pp. 369–374.

[8] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[9] R. Szeliski, "Rapid octree construction from image sequences," *CVGIP: Image Understanding*, vol. 58, no. 1, pp. 23–32, 1993.

[10] K. Kutulakos and S. Seitz, "A theory of shape by space carving," *Int. J. Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.

[11] Steven M. Seitz and Charles R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *Int. J. Computer Vision*, vol. 35, no. 2, pp. 151–173, 1999.

[12] W. B. Culbertson, T. Malzbender, and G. Slabaugh, "Generalized voxel coloring," in *ICCV Workshop, Vision Algorithms Theory and Practice*. 1999, pp. 100–115, Springer-Verlag LNCS 1883.

[13] P. Eisert, E. Steinbach, and B. Girod, "Multi-hypothesis, volumetric reconstruction of 3-d objects from multiple calibrated camera views," in *ICASSP*, 1999, vol. 6, pp. 3509–3512.

[14] A. Broadhurst, T. W. Drummond, and R. Cipolla, "A probabilistic framework for space carving," in *ICCV*, 2001, vol. 1, pp. 388–393.

[15] C. R. Dyer, "Volumetric scene reconstruction from multiple views," in *Foundations of Image Understanding*, L. S. Davis, Ed., pp. 469–489. Kluwer, 2001.

[16] G. Slabaugh, W. B. Culbertson, T. Malzbender, and R. Schafer, "A survey of volumetric scene reconstruction methods from photographs," in *Volume Graphics 2001, Proc. of Joint IEEE TCVG and Eurographics Workshop*, K. Mueller and A. Kaufman, Eds. 2001, pp. 81–100, Springer Computer Science.

[17] O. Faugeras and R. Keriven, "Variational principles, surface evolution, pde's, level set methods and the stereo problem," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 336–344, 1998.

[18] J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered depth images," in *SIGGRAPH*, 1998, pp. 231–242.