Graphical Models **62,** 165–201 (2000)
doi:10.1006/gmod.1999.0519, available online at http://www.idealibrary.com on IDEAL®

Surface Reconstruction and Display from Range and Color Data

Kari Pulli

Nokia Mobile Phones Research, Oulu, Finland
E-mail: kari.pulli@nokia.com

and

Linda G. Shapiro

University of Washington, Seattle, Washington
E-mail: shapiro@cs.washington.edu

Received August 7, 1998; revised August 6, 1999; accepted October 18, 1999

This paper addresses the problem of scanning both the color and geometry of real objects and displaying realistic images of the scanned objects from arbitrary viewpoints. We describe a complete system that uses a stereo camera setup with active lighting to scan the object surface geometry and color. Scans expressed in sensor coordinates are registered into a single object-centered coordinate system by aligning both the color and geometry where the scans overlap. The range data are integrated into a surface model using a robust hierarchical space carving method. The fit of the resulting approximate mesh to data is improved and the mesh structure is simplified using mesh optimization methods. In addition, a method for view-dependent texturing of the reconstructed surfaces is described. The method projects the color data from the input images onto the surface model and blends the various images depending on the location of the viewpoint and other factors such as surface orientation.

1. INTRODUCTION

This paper presents a set of algorithms that together constitute a complete system for scanning and displaying textured objects. The tasks required to fulfill our goal can be organized into a sequence of stages that process the input data. Each of these stages presents us with its own problems. Specifically, the problems addressed in this paper are as follows:

- How can we scan the object surface geometry using color cameras and structured light?

1524-0703/00 $35.00

- Our scanner allows us to digitize only one view of an object at a time so that several scans from many viewpoints are required to scan the whole object. How can we accurately transform all the data in a single object-centered coordinate system?
- How can we convert separate range views into a single surface description? How can we use our knowledge of the scanning procedure to make this process more robust and reliable?
- How can we manipulate the surface description so that it better approximates the input data, and how can we control the size and resolution of the surface representation?
- How can we combine the color information from the different views together with the surface geometry to render realistic images of our model from arbitrary viewpoints?

1.1. *Previous Work*

In this section we briefly mention some previous work in surface modeling from range data and in image-based rendering that heavily influenced our research. Further descriptions of related work can be found in later sections.

1.1.1. Surfaces from range. Surface reconstruction from range data has been studied for a long time. In his dissertation, Besl [1] did an excellent job of exploring low-level range image processing and segmentation. His later work in registration of range data has been influential [2].

Hoppe *et al.* [26–28] developed a three-phase system for reconstructing arbitrary surfaces from unstructured 3D point data. The first phase creates an approximate triangle mesh by extracting the zero set of a signed distance function that was estimated from the point data. The second phase iteratively improves the fit of the mesh and simplifies the mesh. In the third phase the surface representation is changed to Loop's subdivision scheme [34] (extended to allow sharp features), which can represent curved surfaces more accurately and more compactly than a triangle mesh. Around the same time, Chen and Medioni [7, 8] worked on similar problems. Their research addressed registration of range maps into a single coordinate system and subsequent integration of the data into the surface representation.

Curless and Levoy developed two important techniques for building complex and accurate models from range images: spacetime analysis, a robust and accurate scheme for estimating depth using a light stripe scanning system [11], and a volumetric space carving technique for aggregating several range scans into a single geometric model [12].

1.1.2. Image-based rendering. Our work on view-dependent texturing was heavily influenced by the image-based rendering papers published in SIGGRAPH 96. The Microsoft Lumigraph [22] and Stanford Light Field Rendering [32] address the same problem: given a collection of color images of an object from known viewpoints, how can one create new images of the same object from an arbitrary viewpoint? Their solution was to think of the pixels of the input images as half-rays ending at the camera image plane and encoding the apparent directional radiance of the object surface. From these samples a function that returns a color for a directed ray can be calculated. New images can then be created by evaluating that function over the rays associated with each pixel in the output image and painting the pixels with the returned colors. This approach is quite general and allows realistic rendering of objects that are hard to model and render using traditional graphics methods. However, a very large set of input images is required, and the 4D function mapping rays to colors requires large storage.

Debevec *et al.* [15] described a system that allows a user to interactively reconstruct architectural scenes from color images. After the user specifies the basic structure of the geometric models and marks correspondences between image features and model features, the system calculates the actual dimensions of the model elements. The models are then view-dependently textured using the original camera images.

1.2. *Contributions*

The contributions in this paper include the following:

- We have constructed a practical range scanner that uses inexpensive color cameras and a controlled light source. The range data are inferred from camera images using optical triangulation. The quality of the range data is increased by adapting spacetime analysis [11] for our scanner.
- We have developed a new method for pairwise registration of range and color scans. Our method directly addresses the most difficult part of 3D registration: establishing reliable point correspondences between the two scans. We implement a robust optimization method for aligning the views using the established point pairs.
- We have developed a method for simultaneously registering multiple range maps into a single coordinate system. Our method can be used in conjunction with any pairwise registration method. The scans are first registered pairwise. The results of pairwise registration create constraints that can then be used to simultaneously find a global registration.
- We have developed a simple and efficient hierarchical space carving method for creating approximate meshes from registered range maps.
- We have developed a view-dependent texturing method that allows us to combine the color data with reconstructed surface models at display time.

1.3. *Overview*

The first three sections of this paper discuss methods for reconstructing geometric surface models from range scans. Section 2 deals with data acquisition, Section 3 addresses registration of the scan data, and Section 4 discusses methods for reconstructing surfaces from registered range data. Section 5 is concerned with the use of color data and presents our method for view-dependent texturing of scanned surfaces. Section 6 concludes the paper. Most of the material in Sections 2, 4, and 5 has been previously published in various conferences [46–48], while the material in Section 3 has only appeared in a dissertation [45].

2. DATA ACQUISITION

We built our own system for scanning both range and color data. In this section we first describe the hardware configuration of our scanner. We then discuss our simple but robust method for obtaining dense range data from stereo with active lighting. This section concludes with a description of how spacetime analysis, a method that was developed for another type of scanner, was adapted to our system for more reliable and accurate scanning.

2.1. *Scanning Setup*

Our scanning system consists of the following main parts (see Fig. 1). Four NTSC color video cameras are mounted on an aluminum bar. The cameras, which produce images

FIG. 1. The scanner hardware: four video cameras, a slide projector on a turntable, and table lamps.

at 640×480 resolution, are connected to a digitizing board, which can switch between the four inputs under computer control. Below the cameras, a slide projector sits on a computer-controlled turntable. The slide projector emits a vertical stripe of white light, which is manually focused to the working volume. A few lamps provide adjustable lighting for capturing color images.

We obtain range data by active stereo, in our case by sweeping a vertical beam of light over a scene that is kept otherwise dark. One of the cameras is chosen as the base camera, and we obtain a dense range map corresponding to the surfaces visible to that camera. Finally, the lights are turned on, and we take a color image from the base camera. Additionally, we determine background pixels by backlighting objects and tracking which pixels change color and intensity. We will now describe our range through triangulation method.

2.2. *Range from Stereo*

In traditional stereo vision one tries to match pixels in one camera image to pixels in the image of another camera. The pixel matching relies on intensity variations due to surface texture. If the cameras have been calibrated and the matched pixels correspond to the same point on some surface, it is trivial to calculate an estimate of the 3D coordinates of that point.

In our system we solve the pixel correspondence problem by projecting only a single vertical light stripe onto the scene. This is illustrated for the case of two cameras in Fig. 2. The set of points projecting to a pixel forms a line in 3D, and we can calculate that line from calibration data. We match the stripe pixels by projecting the line corresponding to a pixel in the left image onto the right image, and then intersecting the projected line with the stripe in the right image. Once we have matched two pixels, we use the intersection of their corresponding 3D lines to determine the 3D coordinates of the surface point visible through the pixel in the left image. The whole scene is scanned by aiming the stripe at the left side of the working volume, reconstructing surface points illuminated by the beam, turning the beam to the right by a small fixed angle, and repeating the process until the beam has swept over the whole working volume. The result is a dense range map, an image that stores the 3D coordinates of the first visible surface point through each pixel in the left camera image if that point is also visible both to the light projector and the right camera.

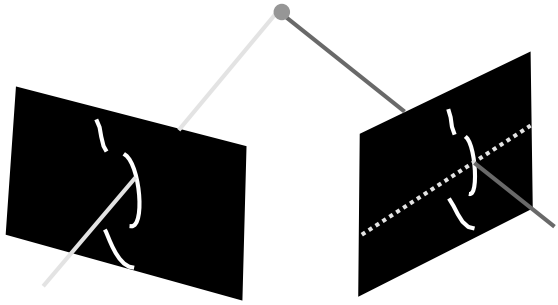


FIG. 2. The epipolar line of an illuminated pixel in the left image is intersected with the stripe in the right image, producing a match and enabling triangulation.

We can increase the reliability of our system by using more than two cameras. In our case, we use four cameras, and we are satisfied if the surface point is visible from the base camera, the light projector, and any of the other cameras. In the case when the point is visible from several cameras, we can check whether the match was reliable by calculating the 3D coordinates of that point from several camera pairs and checking whether the answers agree. The averaged solution from the pairs deemed reliable is finally returned.

2.3. *Spacetime Analysis*

Our scanning method requires that we accurately locate the center of the stripe in a camera image. At first this seems quite easy; techniques such as taking an average weighted by pixel intensities across the beam should give good results. However, there are the hidden assumptions that the surface illuminated by the beam is locally planar and that the whole width of the stripe is visible to the camera.

2.3.1. Problems in locating the stripe. Curless and Levoy [11] noticed that when the assumptions of a planar surface and a fully visible stripe are violated, the range estimates become systematically distorted. They used a Cyberware scanner, which scans range data using a calibrated laser beam with a single calibrated camera. Figure 3 is adapted from [11] to a system of two calibrated cameras and an uncalibrated light beam, and it illustrates several configurations that lead to an incorrect estimate for the surface point coordinates. Figure 3a shows the ideal situation: a completely visible stripe on a flat surface, yielding an accurate range estimate. In Fig. 3b part of the beam is occluded in the right camera by a protuberance of the object. The estimate of the stripe center on the right camera is biased to the left, pulling the range estimate closer toward the left camera. In Fig. 3c the center of the beam is aimed at a sharp silhouette corner of the object, and only half of the beam hits the surface. The stripe center estimate of both cameras is biased to the left, giving incorrect 3D coordinates for the surface point on the silhouette edge. Because of this phenomenon, it is hard to scan the surface reliably all the way to the silhouette boundary. Figure 3d shows the beam pointing at a sharp crease on the surface. The left half of the stripe is fully visible to the left camera, whereas the right half of the stripe is foreshortened because the surface turns away. This results in a bias to the left in the stripe center estimate. By a similar argument, the stripe center is biased to the right in the right camera, resulting in error in scanning the surface point.

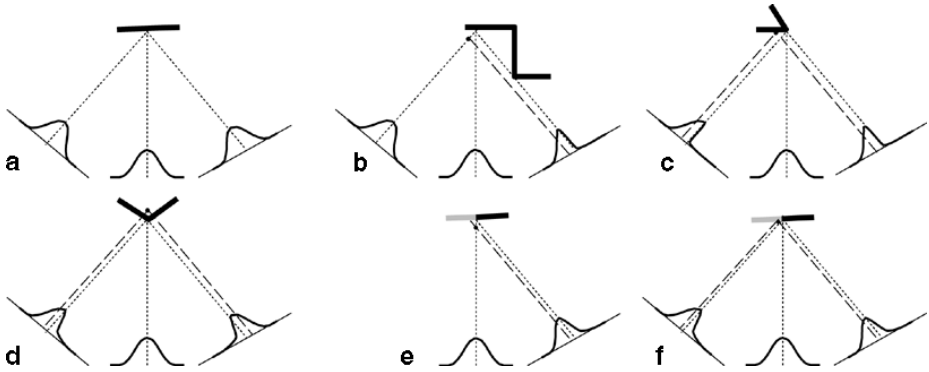


FIG. 3. Error sources in range by triangulation using a light beam with unimodal intensity distribution (center) with two sensors. (a) Ideal situation, a planar object, leads to correct results. (b) A self-occlusion biases the estimate forward. (c) The beam hits the surface only partially, pulling the surface estimate away from the silhouette. (d) Images of beams are partially foreshortened. (e) Intensity changes lead to incorrect surface point estimate in a system using calibrated light source. (f) Intensity changes move the surface point estimate along the surface in a system using calibrated cameras and uncalibrated light source.

It is worth noting that Curless and Levoy presented a fourth case, where changes in surface intensity bias the stripe center estimate, leading to an error in the range estimate (see Fig. 3e). This is not really a problem in our system. Recall that while they use a calibrated light beam and a calibrated camera, we use an uncalibrated light beam and several calibrated cameras. In the case of a planar textured surface, the stripe center estimates would be similarly biased in both cameras. However, as shown in Fig. 3f, the reconstructed point still lies on the surface.

2.3.2. Solution to the problem. The problems illustrated in Fig. 3 can be overcome by reformulating the task. Instead of trying to find *where* the center of the beam is in a single image, one can track the time-varying intensity profiles of the pixels to find out *when* the beam was centered at each pixel. Curless and Levoy [11] call this approach spacetime analysis. Let us assume that the beam is wide enough to cover several pixels in the image, its intensity has a unimodal distribution, and we move the beam in steps that are small compared to the beam width. When the beam approaches the surface point visible through a pixel, the intensity of the pixel first increases and then begins to decrease as the beam sweeps past the point. The shape of the time-varying intensity profile of the pixel has the same shape as the profile of the beam, and the time when the beam was centered on the pixel can be reliably estimated from that profile.

Spacetime analysis produces a function describing when (if ever) the beam was centered on each image pixel. We can use this function in our triangulation method as follows. Choose a pixel in the left image and note the time when the beam was centered at that pixel. Find the epipolar line corresponding to that pixel on the right image as before; i.e., project the 3D line corresponding to the pixel onto the right image. The epipolar line defines a new function that maps a position on the line to a time when the beam was centered on the pixel under the line. To match the original pixel on the left image we simply search for a line position such that the pixel under it has the same time as the original pixel. The 3D coordinates of the surface point are found by triangulation as before. We implemented this method and thereby reduced the errors illustrated in Fig. 3.

3. REGISTRATION

An important step in surface modeling from range data is to register several partial range scans into a common coordinate system. The individual range scans are initially expressed in the sensor coordinate system. An initial registration is often obtained directly from the scanning system or interactively from the user, and the registration is refined by accurately aligning the overlapping parts of the data. If the surface color is scanned along with geometry, the color information can be used to better match data points corresponding to the same point on the object surface. When the color scans are combined to form texture maps that are then applied to the reconstructed model, it is important that the colors also align precisely. Aligning only the geometry may lead to blurring and ghosting in textures.

Registration can be subdivided into the correspondence and alignment subproblems. The *correspondence problem* is to determine for a given point in one scan the points in the other scans that correspond to the same point on the object surface. The *alignment problem* is to find a rigid 3D transformation that aligns the corresponding points. Of these two problems the correspondence problem is the harder one, while closed form solutions exist for determining the alignment in the least-squares sense.

Most registration methods use heuristics to determine approximate point correspondences and improve the alignment, iterating these two steps until the process converges. We will cover previous approaches for correspondence heuristics and alignment methods in Section 3.1. These methods were typically first developed for pairwise registration of range scans; we will also summarize how they have been extended into multiview registration. In Section 3.2 we present a new method for determining a consistent set of point correspondences that effectively uses and aligns the color information. In Section 3.3 we present a robust method for aligning range scans given a set of point correspondences. The method first finds a transformation that aligns most of the paired points, then automatically removes bad pairs (false matches) and refines the alignment using all the remaining pairs. In Section 3.4 we introduce a fast and robust approach for multiview registration that completely separates correspondence determination and simultaneous alignment of all the views.

3.1. *Background and Related Work*

If we have two overlapping range scans of the same object, we can register the views by aligning their overlapping geometry. A good registration can be found by minimizing the equation

$$\sum_{i=1}^{N_p} \|\mathbf{T}\mathbf{p}_i - \mathbf{q}_i\|^2, \qquad \mathbf{q}_i = \arg \min_{\mathbf{q}\in\mathbf{Q}} \|\mathbf{T}\mathbf{p}_i - \mathbf{q}\|. \tag{1}$$

Here we are looking for a rigid 3D transformation $\mathbf{T}$ that minimizes the distances of the scanned points $\mathbf{p}_i \in \mathbf{P}$ to a set $\mathbf{Q}$, which can be another set of scanned points, a surface fitted to a point set, or a complete surface model. Equation (1) is difficult to solve directly, as it tries to simultaneously solve the correspondence (find $\mathbf{q}_i \in \mathbf{Q}$ corresponding to $\mathbf{p}_i$, and if none does, exclude the ith element from the sum) and the alignment (find transformation $\mathbf{T}$). However, since it is easy to solve the alignment given the correspondences, and vice versa, the idea of iterating over two simpler subproblems is used in most practical registration methods.

Besl and McKay [2], Chen and Medioni [8], and Zhang [63] proposed minimizing Eq. (1) by iterating between pairing corresponding points and solving the transformation. The objective function in the iterative closest point (ICP) approach is similar to Eq. (1):

$$\sum_{i=1}^{N_p} \|\mathbf{T}_k\mathbf{p}_i - \mathbf{q}_i\|^2, \qquad \mathbf{q}_i = \text{Match}(\mathbf{T}_{k-1}\mathbf{p}_i, \mathbf{Q}). \tag{2}$$

The estimate of the registration transformation $\mathbf{T}$ is refined in small steps. First, while the transformation is held fixed, the points $\mathbf{T}_{k-1}\mathbf{p}_i$ are paired with $\mathbf{q}_i$ using a correspondence heuristic. Now the pairs are held fixed, and a transformation $\mathbf{T}_k$ that brings $\mathbf{P}$ closer to $\mathbf{Q}$ is calculated. As the surfaces move closer, finding better point pairs becomes easier and more reliable, which enables finding better transformations, so pairings and alignments can be iterated until the process converges.

3.1.1. Correspondence heuristics. Several methods have been suggested for pairing points and for determining whether some points should be paired at all. Potmesil's [44] approach was to estimate a local normal vector at the point $\mathbf{p}_i$ and to pair it with the intersection of the normal and the other surface $\mathbf{Q}$. If the normal did not intersect $\mathbf{Q}$, $\mathbf{p}_i$ was ignored. Chen and Medioni [8] adopted the same strategy.

Besl and McKay [2] and Zhang [63] paired each point $\mathbf{p}_i$ with the closest point in $\mathbf{Q}$. Godin *et al.* [21] looked for the closest compatible point, where the points were compatible if their colors are similar enough. Gagnon *et al.* [19] found the closest point such that its normal vector does not point away from the first point. One can also modify the distance function so that it takes into account not only the physical distance but also the difference in the associated information (Eggert *et al.* [17] compared normal vectors).

These heuristics still produce many pairs that are unlikely to correspond even approximately to the same surface location. Methods have been developed to disallow or cull some such pairs. Turk and Levoy [59] eliminated pairs where the points are too far apart, or where the closest point is on the silhouette of the range map. Dorai *et al.* [16] discarded pairs if they were not compatible with the neighboring pairs. Take two pairs, $(\mathbf{p}_i, \mathbf{q}_i)$ and $(\mathbf{p}_j, \mathbf{q}_j)$. Unless the distance $\|\mathbf{p}_i - \mathbf{p}_j\|$ equals the distance $\|\mathbf{q}_i - \mathbf{q}_j\|$, it is not possible that both pairs are valid at the same time, since a rigid registration transformation preserves 3D distances.

Weik [61] developed a method that avoids the closest point search by projecting points from one range map to another range map. As illustrated in Fig. 4, a point $\mathbf{p}$ is paired by projecting it to $\mathbf{p}'$ on the image plane of the scanner of the other view and finding the point $\mathbf{q}'$ in the other view that projects to the same location. However, he also used image intensity information to find a better pair for a point. The intensity difference $\Delta I = I_0(\mathbf{p}) - I_1(\mathbf{p}')$, intensity gradient $\nabla I_1(\mathbf{p}')$, and displacement $\mathbf{d}$ such that $\Delta I = \nabla I_1(\mathbf{p}') \cdot \mathbf{d}$ are calculated. The original point is paired with the point $\mathbf{q}$, which projects to the new corrected image location $\mathbf{p}' + \mathbf{d}$. Weik also prevents some points from being paired if they cannot possibly be visible, due to self-occlusion in the direction of the other viewpoint (point $\mathbf{x}$ in Fig. 4).

A few researchers [21, 63] have noted that the point pairing should be symmetric. The pairing can easily be made symmetric by first projecting points in $\mathbf{P}$ to $\mathbf{Q}$, then projecting points in $\mathbf{Q}$ to $\mathbf{P}$, and finally merging the two sets of pairs.

3.1.2. Alignment methods. The most widely used alignment method finds the rigid Euclidean motion that minimizes the sum of squared distances between the paired points.

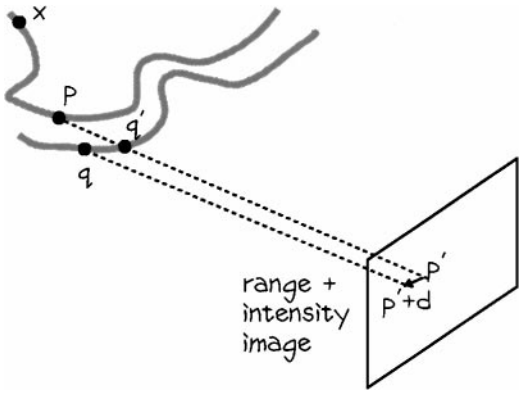


FIG. 4. Project a point to the other view's image, apply an image gradient-based correction, and project back to the other view's range data.

Closed form solutions were published independently by Horn [29] and Faugeras and Hebert [18], and Horn's method was later reformulated by Wang and Jepson [60].

This approach works well if the paired points really correspond to the same points on the surface. Typically, however, even a few bad point pairs pull the solution away from the perfect registration, producing a situation like the one illustrated in Fig. 5. The surfaces are in a fair registration, but they still need to slide a bit past each other. Several pairs pull the solution to the correct direction, but most paired points are very close to each other and therefore resist motion in any direction, preventing the surfaces from moving more than a very small step, which slows the convergence near the solution.

Chen and Medioni [8] moved a surface so that points on it move closer to the tangent planes of their counterparts on the other surface (Fig. 6). Whereas in the previous method ideal springs were attached to point pairs, here only the first end of the spring is fixed, while the other end is free to slide along the other surface. No penalty is associated with surfaces sliding past each other, so larger steps are possible, and the registration is likely to converge faster. Although the sliding spring approach typically converges faster, there are situations where the fixed spring approach works better. If the point pairs are good, the fixed springs pull the surfaces to alignment immediately, while with sliding springs a good pairing may not move the surfaces at all if the points already lie in the tangent plane of their counterparts.

Champleboux *et al.* [4] designed a function that can be quickly evaluated at any 3D location and which returns the distance from that point to the surface. The function representation allows speedy numerical estimation of its gradients, and a surface is moved closer to the other using a gradient descent type of optimization.

FIG. 5. If many of the paired points do not correspond to the same surface points, many seemingly good pairs (short distances) resist motion in all directions.

FIG. 6. Minimizing the distance from a point to a plane allows the left pair to pull the surfaces into alignment without the other pairs resisting tangential sliding.

3.1.3. Other approaches to registration. Higuchi *et al.* [23] used a representation that decouples rotation and translation and noted that solving the translation is easy once the relative rotation has been solved. A regular spherical mesh is fitted to the data so that the edges have approximately the same length. At each vertex, a discrete estimation of the Gaussian curvature of the underlying surface is calculated. The rotation can be solved by searching for the rotation of a sphere that minimizes the sum of squared differences of the curvature estimates. This method can be applied only to objects topologically equivalent to a sphere.

Blais and Levine [3] evaluated the quality of registration by projecting one range map to the image plane of the other. This technique was later used by Masuda and Yokoya, and Weik. The projection is used only for evaluation, not for registration. The optimization is done using simulated reannealing, a stochastic global optimization method.

3.1.4. Multiview registration. A simple method for registering several views is by registering views sequentially: the unregistered views are registered, one at a time, to a view that has already been registered. The problem with this approach is illustrated in Fig. 7. When the views are registered pairwise in the scanning order, small registration errors begin to accumulate.

Turk and Levoy's [59] solution to the compounding registration error relies on their scanner's ability to take cylindrical scans. A single cylindrical scan of the object is taken and is used as an anchor; other scans are then registered with the cylindrical scan. Chen and Medioni [8] registered the views one at a time, merging the registered view into a single "metaview." All the new views were then registered with the metaview rather than just a single neighboring or anchor view. This approach was also taken by Masuda *et al.* [37].

Gagnon *et al.* [19] pointed out that when more views are added, it is possible that they bring information that could improve the registration of the previously registered views.

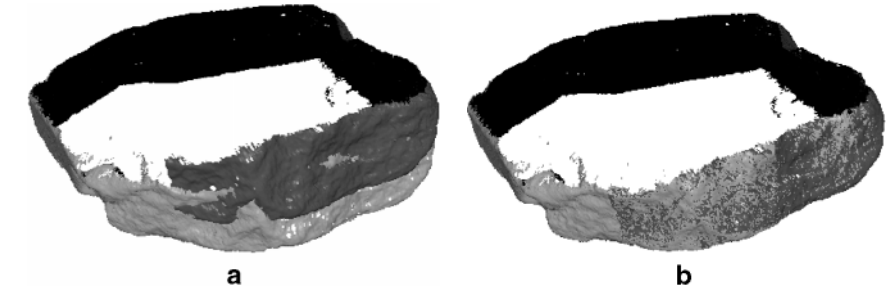


FIG. 7. (a) Small errors accumulate, resulting in a large gap. (b) Multiview registration enforces a consensus solution.

They project a set of points in view i to all the other views and calculate a transformation that registers view i using all the paired points. This process is iterated repeatedly over all the views until the registration slowly converges. Jin *et al.* [30] tried to solve the same problem by incrementally building a surface model, against which new views can be registered and already-registered views can be reregistered, but the method was very slow.

Eggert *et al.* [17] attempted to overcome the thresholding difficulties of pairwise correspondences (maximum allowed difference in position, normals, color, etc.) that most previous methods have to deal with. They assume that all points on the object's surface have been observed at least twice. Therefore, a proper corresponding location exists for each point; it just needs to be found. They simply pair each point with the closest compatible point. However, we have observed that pairing a point in only one other view can prevent the algorithm from converging to a correct solution. Sometimes the views form cliques such that all the views belonging to a clique are in a good registration, but the cliques themselves are not well registered to each other. As a result, the closest point is usually found in another view of the same clique, and no further progress is made, even though the global registration may be far from perfect.

3.2. *Correspondence Problem*

The traditional correspondence heuristics pair points using simple local pairing rules. Figure 8 shows two examples where a surface is paired with a copy of itself that has been translated to the left and upward. In Fig. 8a points are paired with the closest point with matching normal, while in Fig. 8b the heuristic looks for the closest point with matching color. Not only are the pairings produced using local heuristics inconsistent, but almost none of the paired points actually correspond to the same point on the surface. It is clearly not enough to just throw away bad pairs; we should instead concentrate on finding good ones.

Before suggesting an alternative point pairing method, we list some desirable properties of good pairings:

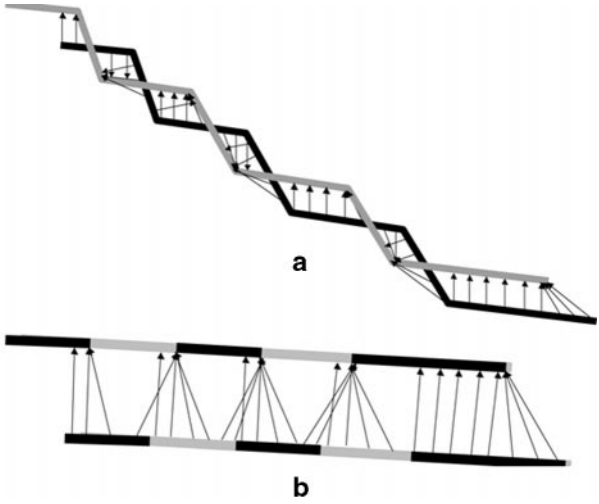


FIG. 8. Local heuristics lead to inconsistent pairings. (a) Points are paired with the closest point with compatible normal vector. (b) Points are paired with the closest point with compatible color.

- Each point and its partner should correspond to the same location on the surface, and only points from surfaces visible in both views should be paired.
- The mapping within the overlapping surface should be bijective, i.e., one-to-one (a point can be paired with only a single point) and onto (every point in the overlapping region should have a partner).
- The pairs should be consistently related with a single rigid 3D motion.
- The color data should be aligned as well as the geometry. In fact, surfaces that have rotational or translational symmetries cannot be unambiguously registered using geometric data alone [19, 61].

It is difficult to create a pairing with the above properties using only local heuristics that myopically look for a closest compatible point without considering how the other points are paired. We propose a new, more global metric for simultaneously pairing the points on overlapping surfaces. The basic idea is to align images of the data sets and pair surface points projecting to the same image location.

Consider two textured surfaces that are already in close alignment. If the scanned surfaces are rendered as they would be seen from an arbitrary viewpoint (from which the overlap of the two surfaces is visible) the resulting 2D color images are also in alignment. Each point on surface A projects to the same pixel as its corresponding point on surface B.

In our case we have two views of a textured surface in rough alignment, and we would like to improve their registration. If we could move one of the views (partial surface) such that its projected image aligns well with the image of the other surface, we could be fairly confident that visible surface points projecting to the same pixel correspond to the same point on the object surface. We can then find good point pairs by pairing points that project to the same pixel. A brief examination confirms that all the properties that we listed as desirable are indeed satisfied by this algorithm.

Color Plate 1 illustrates the basic idea in our point-pairing mechanism. We have two scans of color and range data, and the range data are roughly aligned. A textured mesh of one scan is projected onto the color image of the other scan (left images) and moved in 3D so its projection aligns with the color image (right images). A dense set of 3D point pairs is obtained by pairing points projecting to the same pixel. We now discuss our point-pairing method in more detail.

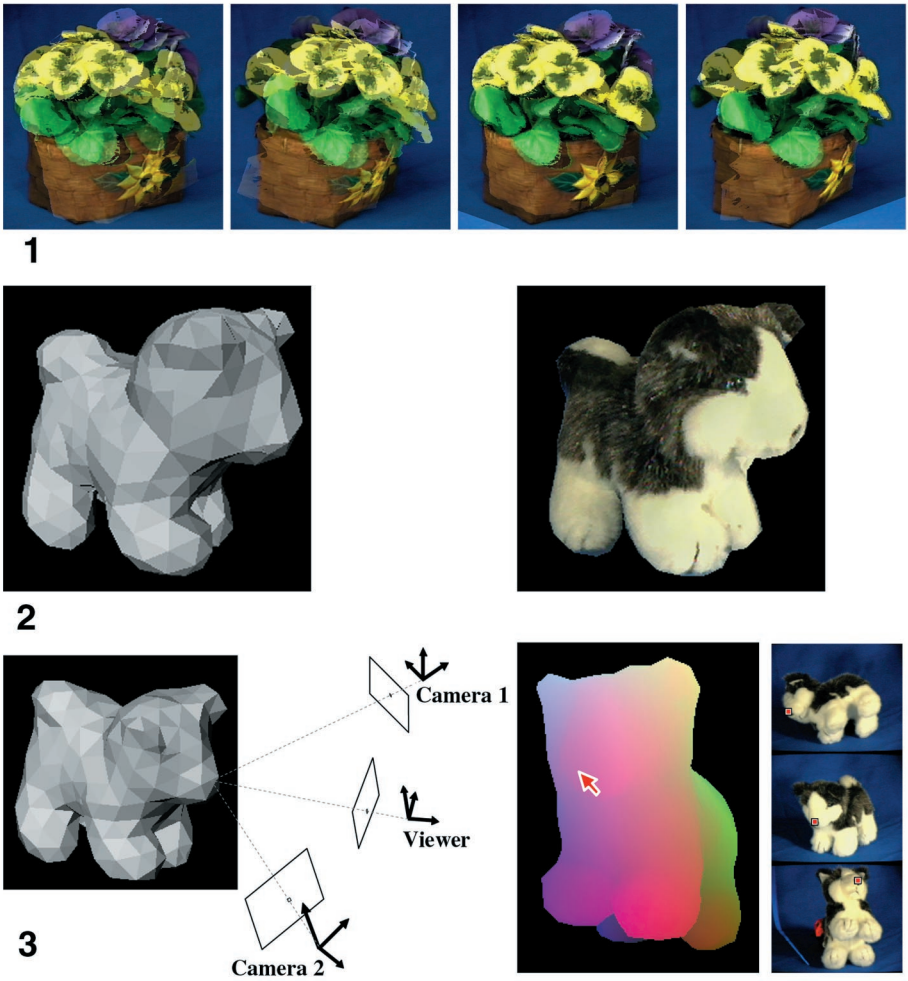
3.2.1. Color image alignment. We have color and range scans from two different viewpoints, and we want to align the projections of the color information on the first camera's image plane. We do this by projecting the textured surface of the second scan on the first color image, calculating color differences and gradients, linearizing the problem for fast implementation, and converting the differences to incremental 3D motions that better align the images. Note that this requires a rough initial alignment so that the projected second scan overlaps at least partially with the first scan, and the orientations do not differ more than 20–30°.

We model the imaging geometry as follows (see Fig. 9). The first camera with focal length $\mathbf{f}$ is located at $\mathbf{x}_0$ and oriented along $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ such that the viewing direction is $-\mathbf{e}_3$. The images are assumed to already be in a rough alignment and we move the second surface by rotating it around its center of mass $\mathbf{c}$ by the angles α, β, and γ around the $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ axes, respectively, followed by a translation $\mathbf{b}$. In addition to the color data, the second surface projects a height field $\mathbf{h}$ onto the image plane of the first camera.

We want to minimize

$$E(\mathbf{T}) = \sum_{(u,v)} [\mathbf{w}_{(u,v)}(I_1(\psi(u, v)) - I_0(u, v))]^2. \quad (3)$$

In this equation ψ is the inverse of the image flow telling which pixel moves to (u, v). The image flow is induced by a 3D transformation $\mathbf{T}$, which refines our initial rough registration until no small additional motion can reduce the error of Eq. (3). I_0 is the original color



COLOR PLATE 1. Consistent pairing of points. Roughly aligned textured meshes are projected on color images (left) and moved so their projections align with the images (right).

COLOR PLATE 2. The geometry of a toy husky (left). Realism is greatly enhanced by texture mapping (right).

COLOR PLATE 3. (a) A ray from the viewer's center is cast through a pixel, intersecting the object surface. The intersection point is projected back to color images, producing candidate pixels for coloring the pixel of the original ray in the viewer. (b) A false color rendering of the surface geometry uses hardware to find the surface point visible through each pixel. Red, blue, and green channels encode x, y, and z coordinates, respectively. (c) The 3D point corresponding to the pixel pointed to in (b) is projected into three color images (the dot on the nose).

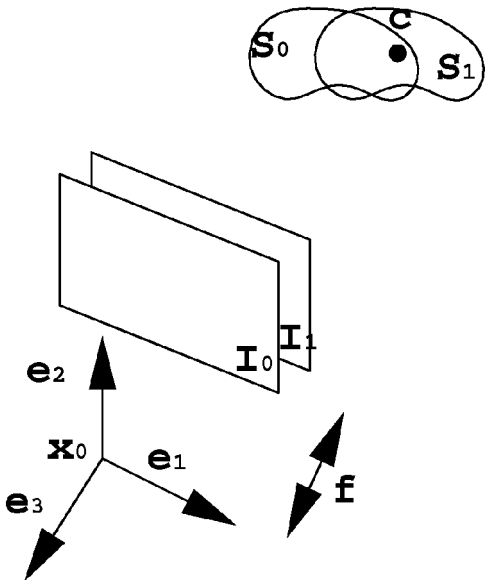


FIG. 9. Color image alignment setup.

image, I_1 is the image of the second surface, and $\mathbf{w}_{(u,v)}$ is a per pixel weight. We solve this iteratively using the Gauss–Newton method, that is, by doing the Taylor series expansion of $I_1(\psi(u, v))$ around identity $\mathbf{T}$, truncating the series, and solving for the small transformation $\dot{\mathbf{T}}$, yielding

$$\sum_{(u,v)} \left[\mathbf{w}_{(u,v)} \left(\mathbf{g}^T_{(u,v)} \dot{\psi}(u, v)\dot{\mathbf{T}} + \mathbf{e}_{(u,v)}\right)\right]^2 = 0. \tag{4}$$

In this equation $\mathbf{g}^T_{(u,v)} = \nabla I_1(u, v)$ is the image gradient, and $\mathbf{e}_{(u,v)} = I_1(u, v) - I_0(u, v)$ is the color error. The remaining terms are

$$\dot{\psi}(u, v)\dot{\mathbf{T}} = \frac{1}{\mathbf{h}(u, v)} \begin{bmatrix} \mathbf{f}\mathbf{e}_1 - u\mathbf{e}_3 \\ \mathbf{f}\mathbf{e}_2 - v\mathbf{e}_3 \end{bmatrix} \begin{bmatrix} 0 & Z & -Y & 1 & 0 & 0 \\ -Z & 0 & X & 0 & 1 & 0 \\ Y & -X & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \dot{\mathrm{b}}_0 \\ \dot{\mathrm{b}}_1 \\ \dot{\mathrm{b}}_2 \end{bmatrix}, \tag{5}$$

where $[X \;\; Y \;\; Z]^T$ is the vector from the center of rotation to the surface point projecting to image location (u, v):

$$[X \quad Y \quad Z]^T = \mathbf{P}(u, v) - \mathbf{c} = \mathbf{x}_0 + \frac{\mathbf{h}(u, v)}{\mathbf{f}}(u\mathbf{e}_1 + v\mathbf{e}_2 + \mathbf{f}\mathbf{e}_3) - \mathbf{c}. \tag{6}$$

We solve this from the normal equations $\mathbf{A}\dot{\mathbf{T}} = -\mathbf{b}$ where $\mathbf{A} = \sum_{(u,v)} \mathbf{w}_{(u,v)} \dot{\psi}(u, v)^T \mathbf{g}_{(u,v)}\mathbf{g}^T_{(u,v)}\dot{\psi}(u, v)$ is an 8×8 matrix and $\mathbf{b} = \sum_{(u,v)} \mathbf{w}_{(u,v)}\mathbf{e}_{(u,v)}\dot{\psi}(u, v)^T \mathbf{g}_{(u,v)}$ is an 8-vector. $\dot{\mathbf{T}}$ is solved best by applying Cholesky decomposition for positive semidefinite matrices to $\mathbf{A}$.

3.2.2. Practical issues. The colored surfaces are created by texturing each range image with its associated color image. The range images are converted to meshes by connecting the range maps into dense triangle meshes. We use a heuristic that takes into account both triangle edge lengths and triangle orientation and removes triangles that are likely to connect two surfaces across a step edge due to self-occlusion. The dense meshes are simplified to about 10,000 triangles [20].

In order to calculate $\dot{\mathbf{T}}$, we need to come up with values for $I_1(u, v)$, $\mathbf{h}(u, v)$, $\mathbf{g}_{(u,v)}$, and $\mathbf{w}_{(u,v)}$ while the rest are given. The first two we obtain simply by rendering the second colored surface from the viewpoint of the first camera and by consulting the frame and z-buffers, respectively. Simple image processing of $I_1(u, v)$ produces an estimate of $\mathbf{g}_{(u,v)}$. The weights $\mathbf{w}_{(u,v)}$ are determined as follows. Ideally, portions where the surface projects in I_1 are a subset of I_0. We want to evaluate the function only within that subset by setting the weights $\mathbf{w}_{(u,v)}$ to zero for those pixels of I_0, where I_1 does not contain data. In order to make the weight function smooth, the pixels well inside the valid range data in I_1 have weight one, while the pixels closer to boundaries have a smaller weight that smoothly diminishes to zero at the boundary. In our implementation we used a linear feathering ramp whose width was 20 pixels. The ramp distance was calculated from the boundary using 4-connectivity (city-block distance), all pixels outside of the boundary obtain zero weight while all other pixels obtain nonzero weights. The weights can also be used to downweight samples that are suspected of being less accurate for various reasons. The image alignment is performed twice, once from each scanner position. The two sets of images are registered separately, producing two sets of point pairs. This way both views are treated symmetrically, and neither view prevails over the other.

The basic image alignment method only finds a local minimum for the image registration error (Eq. (3)). The implementation becomes both faster and more robust when we solve the problem in a hierarchical fashion. We begin the hierarchical search by building an image pyramid for the input color. We first register low-resolution versions of the images, then increase the resolution, and continue the process until the level of resolution of the input color images is reached. The registration is performed identically at every resolution level.

3.3. *Robust Alignment*

Once the color images are aligned most of the point pairs correspond to almost the same surface point, but we cannot expect that to be the case with all pairs. Points from different surfaces may be matched, for example, next to step edges or when the views fail to scan all the visible surface. If we proceed to minimize the sum of squared pairwise distances, the false pairings will have a large influence on the result. We could try to use various compatibility thresholds to filter out the bad pairs. However, it is far from obvious how to automatically choose the thresholds. Instead, we combine Horn's least-squares point set alignment method with a robust statistical method.

Masuda and Yokoya [38] have previously used a *least median of squares* (LMS) approach [49] to robustly register noisy views when there is at least 50% overlap between the views. In the following, we describe our method based on a *least trimmed squares* (LTS) approach [50], which has a better convergence rate and smoother objective function than LMS. We have also developed a method for determining the percentage of correctly paired points, so they all can contribute to the final result.

3.3.1. Least trimmed squares. LTS [50] can correctly fit a function when up to 50% of the data are contaminated by outliers. In LTS one tries to find a function such that the sum

$$\sum_{i=1}^{h} (r^2)_{i:n} \tag{7}$$

is minimized, where $(r^2)_{1:n} \leq \cdots \leq (r^2)_{n:n}$ are the ordered squared residuals of the function estimates and the measured values, n is the number of data points, and h determines which portion of the data needs to fit well to the function. A typical choice for h, when the rate of outliers in the data is unknown, is $n/2$. In practice LTS is implemented by randomly selecting N_S data points that (over)determine the function, evaluating Eq. (7) with that function, storing the function and evaluation, repeating this N_I times, and choosing the function with the best evaluation. The probability that at least one sampling consists only of inliers is

$$1 - (1 - (1 - \epsilon)^{N_S})^{N_I}, \tag{8}$$

where ϵ is the rate of outliers. Assuming over half of the data are inliers, the inlier errors follow Gaussian distribution, and n is large, one can estimate the standard deviation of the fit from the median of squared residuals [49]:

$$s^0 = 1.4826\sqrt{r^2_{n/2:n}}. \tag{9}$$

In our case the estimated function is the rigid 3D transformation that aligns two range scans. In each iteration, N_S paired points are selected and the 3D motion that aligns the pairs is calculated. The solution is evaluated by applying that motion to the whole data set, pairing all the points, and summing the h smallest squared point distances. If the current solution evaluates better than other tries so far, the solution is stored. Note that the set of point pairs from which N_S pairs are selected does not change for N_I iterations.

3.3.2. Better fit using all the inliers. Having obtained an estimate for the registration using LTS, we try to improve the result by finding a least-squares solution using all good point pairs. We first apply the best LTS solution, create new point pairs, and estimate the standard deviation of the inliers using Eq. (9). A least-squares solution is found by using the pairs that are within 2.5 times the standard deviation of the current fit. In a normally distributed sample population, over 98% of the samples are within 2.5 standard deviations from the mean.

If, for example, 80% of the data are inliers, it is far better to sum the 80% smallest squared residuals when evaluating an LTS trial, rather than only 50% as the basic method advises. While estimating the standard deviation, we also calculate the percentage of the pairs that are inliers. We use this estimate in the next round of LTS trial evaluations. In our implementation, 20 rounds of LTS are performed with five point pairs each time. Assuming that 80% of the paired points correspond to the same surface point, there is a 0.9996 probability of having at least one round in which all the five pairs are inliers. Note that this 80% is over the point pairs that project to the same pixel after color image alignment rather than 80% of all range data. In our experiments the inlier percentage is typically much higher, at least 90%, even though the surfaces have only 30% of overlap.

The 3D transformation estimate of each round is evaluated by rendering the geometry of the meshes into the z-buffer and pairing points that project to the same pixel. No 2D image

alignment is performed. From the viewpoint of I_0, residuals are calculated for all pixels where a mesh from I_1 is projected. If, however, I_0 does not contain valid data on such a pixel, the residual of the pixel is set infinitely large. The residuals are also calculated from projecting I_0 over I_1.

After LTS, a new set of point pairs is formed by rendering the geometries to the z-buffer and pairing points that fall to the same pixel. The standard deviation of the inliers is estimated, and a least-squares solution is found using the point pairs with distance less than 2.5 standard deviations.

3.4. *Multiview Registration*

The methods for multiview registration described in Section 3.1.4. use the basic ICP idea of iteratively pairing points and moving the views closer. One view at a time, points in the current view are paired with points in the other views, the current view is aligned with the others, and the process is iterated until convergence. Point correspondence determination is unreliable when views are not yet in good alignment. But even if accurate correspondences are determined, the registration will not improve very much if the other views are not already in consistent alignment. Therefore a large number of iterations are required, and in each round the point correspondences have to be determined again. Additionally, due to unreliable pairs, the registration may converge to a local, instead of the global, minimum.

We can significantly accelerate multiview registration if we separate the correspondence determination and the finding of a consistent global alignment. First, we find a set of *reliable* point pairs between each view and several neighboring views. The set of pairs is then kept unchanged, and a global registration is determined that simultaneously aligns all the pairs.

We begin by constructing an initial registration via interactive sequential pairwise registration. The user is given visual feedback of the initial registration by being shown the subsampled registered range data, surrounded by thumbnails of the color images at the viewpoint of each sensor, as in Fig. 10. Each view is then registered pairwise with a few neighboring views. It is of no particular importance which pairwise registration method

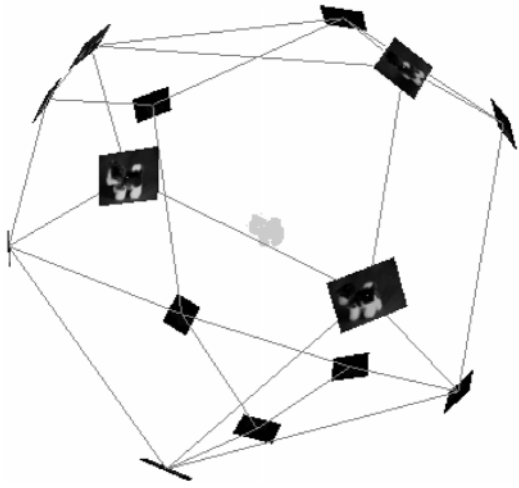


FIG. 10. Multiview registration. A global registration is found from reliable point pairs found through pairwise registration.

is used, as long as the results are accurate and reliable. Once a pair of views are in good registration, reliable point pairs are found and only pairs that are very close to each other are retained. The pairing is done symmetrically from one view to the other.

With the pairwise registrations forming a graph such that every view is connected to every other view through several paths, the global registration is determined using the stored point pairs. Stoddart and Hilton [55] and Eggert *et al.* [17] both developed a method that attaches imaginary springs to the paired points and simulates a simplified version of the resulting dynamic system.

Our approach is to modify a traditional ICP successive relaxation procedure, while keeping the point pairs fixed. One at a time, we add views into a set, and while doing so, we align the new view using only point pairs that connect the view with other views already in the set. Once all the views have been added, the relaxation process begins. A view is simultaneously registered with all the views with which it has point pairs. This process is repeatedly iterated over all the views as long as the registration improves. For data sets consisting of one or two dozen views, the convergence is typically obtained in under a minute. We also verify registration results by aligning the views simultaneously using a conjugate gradients method. The final registration error is comparable, though the execution takes several minutes. Yet even that is dramatically faster than our earlier approach, similar to the one in [19], which took hours because the costly projection step had to be repeated at every iteration, and typically hundreds of iterations were required.

4. SURFACE RECONSTRUCTION

Registration brings separate scans into a single coordinate system, yet the registered scans remain separate until they are processed into a single surface. There are two important steps in surface reconstruction from range data. The data must be *integrated* into a single representation, and surfaces must be *inferred* either from that representation or directly from the original data. There does not seem to be a clear consensus as to the order in which these two steps should be taken.

Several researchers have decided to first approximate each data set as a polygonal mesh [43, 51, 54, 59]; the individual meshes are then connected to form a single mesh covering the whole object. Soucy and Laurendeau [54], for example, first divided the range data into subsets based on which surface regions are visible within each view. In each subset, the redundancy of the views was used to improve the surface approximation. Finally, the triangulated non-overlapping subsets were connected into a single mesh.

Other authors [12, 24, 27] chose first to integrate the data into a signed distance function and then to extract a polygonal mesh using the marching cubes algorithm [35]. Hoppe *et al.*'s [27] method was designed to work with arbitrary point clouds. They first estimated local tangent planes to each data point, and then propagated the tangent plane orientations over the whole data set. The distance of a 3D point is evaluated as the distance to the closest tangent plane, where the distance is positive if the point is above the plane, negative otherwise. Ideally the zero set of the distance function would follow the object surface. However, the local tangent plane estimation phase smooths the signed distance function. Hoppe *et al.* later improved the results by fitting the mesh better to the original data [28].

Curless and Levoy improved the distance function estimation for the case that the input is structured in the form of a set of range maps [12]. They defined a volumetric function based on a weighted average of signed distances to each range image. Their scheme evaluates this

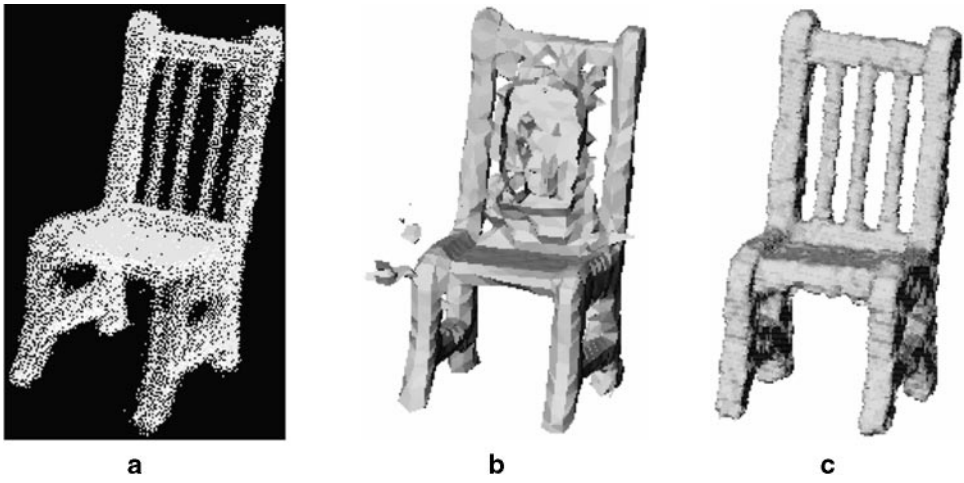


FIG. 11. An example where the old method for obtaining initial surface with correct topology fails. (a) The registered point set (data courtesy of Patrick Flynn of WSU). (b) The result from using the method by Hoppe *et al.* [27]. (c) The result from our method.

volumetric function at discrete points on a uniform 3D grid and uses these discrete samples to determine a surface that approximates the zero set of the volumetric function. Like that of Hoppe *et al.*, their scheme may fail to detect features smaller than the grid spacing and has difficulties with thin features. However, their use of space carving (throwing away regions known to lie outside the object) makes the approach much more robust. The signed distance function is evaluated at a fixed, very fine resolution, and the output of the marching cubes algorithm is used as the final result.

We originally used the method of [27] to obtain an initial mesh, which we then simplified and improved the fit to the original data using the methods of [28]. The initial mesh algorithm does not require any knowledge (such as viewing directions) aside from the data points themselves, and works quite nicely if the data do not contain outliers and uniformly sample the underlying surface. Unfortunately real data often violate both of those requirements. Figure 11a shows the eight registered range maps of a toy chair. Although most of the outliers and background data points were interactively removed from the data sets prior to initial mesh estimation, many outliers remain, especially around the spokes of the back support; some data were missing, and the data were not uniform enough for the algorithm to produce a topologically correct result (see Fig. 11b).

We decided to abandon Hoppe *et al.*'s [27] method and create a more robust method that produced the result shown in Fig. 11c. Like Curless and Levoy, we use the idea of space carving. However, we use a hierarchical approach that saves storage space and execution time, and we do not attempt to directly obtain the final result. Instead, we concentrate on capturing the object topology as correctly as possible given the input data. Additionally, we use only robust methods such as interval analysis [53] that enable us to recover thin objects that are typically difficult to model using the signed distance function approach. We keep the mesh optimization method of [28] since for our purposes we typically want a mesh that is both accurate and concise. Typically one-pass methods such as [12] can only produce meshes that are either dense or not very accurate. Mesh optimization first improves the accuracy and can also simplify the mesh drastically with little sacrifice in accuracy.

In the rest of this section we describe our hierarchical space carving method for determining the object topology and creating initial meshes. We also briefly describe some results of using the mesh optimization algorithm of Hoppe *et al.* [28], and conclude with a discussion of results and various properties of our method.

4.1. *Hierarchical Space Carving*

Our method has the following assumptions. The data are assumed to be a collection of registered range maps acquired from various viewpoints around the object. The data must be obtained using some line-of-sight method, so that every straight line between a data point and the sensor lies entirely outside the object. Finally, the calibration parameters of the sensor must be known in the sense that any 3D point can be projected to the sensor's image plane, or equivalently, the correspondences between 3D lines and pixels on the image plane are known. If the calibration parameters are not known beforehand, it is often possible to estimate them directly from the known correspondences between 3D points and their 2D image coordinates. For example, in the case of the data set in Fig. 11, we did not have the calibration parameters but estimated them using Tsai's method of camera calibration [58].

4.1.1. Space carving. The idea of space carving is illustrated in Fig. 12. In Fig. 12a there is a working volume, and the object we are trying to reconstruct is only known to be somewhere inside of it. Figure 12b shows a scanner and its viewing cone. The left and bottom sides of the object are visible to the scanner; therefore the volume between the scanned surface and the sensor can be removed. If there are data from the background in addition to the object, even more of the unknown volume can be removed. Figure 12c shows how another view carves more space away.

In the end a connected volume that closely approximates the object is left. An approximation of the object surface can be obtained by extracting the boundary between the space that was carved away and the volume that remains. The resulting surface estimate can have arbitrary topology; i.e., the object can have any number of holes, yet the surface itself is without boundaries or holes, even when some of the object surface was not scanned. In general, the resulting surface is the one with maximum volume such that the model is still compatible with all of the scanned data. Figure 12c shows a situation where the whole top side of the object remained unseen by the scanner. Nevertheless, the gap in the input is filled by a plausible solution that is not far from the true surface.

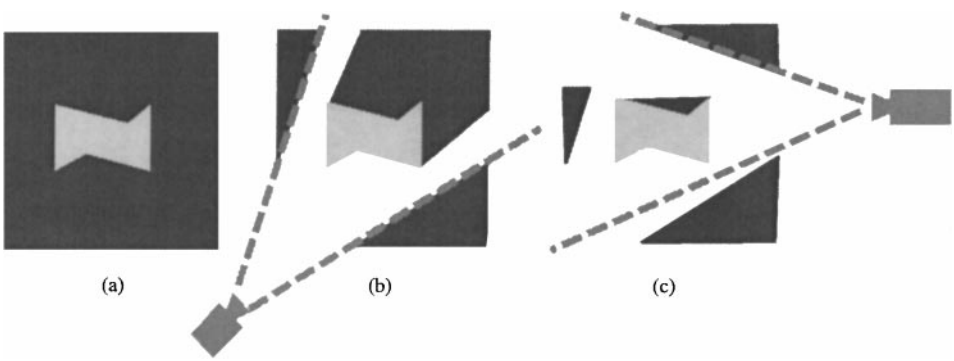


FIG. 12. The idea of space carving. (a) The working volume contains an object. (b) A scan can prove some of the volume lies outside the object. (c) More scans remove more space, but the object remains.

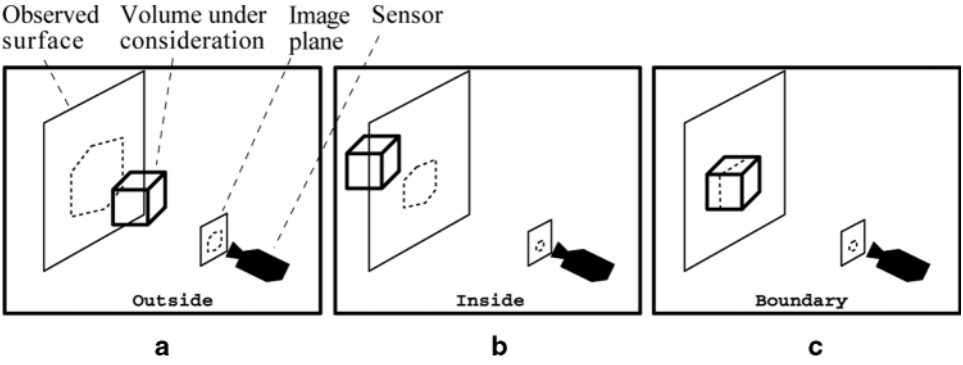


FIG. 13. The three cases of the algorithm. (a) The cube is in front of the range data. (b) The cube is entirely behind the surface with respect to the sensor. (c) The cube intersects the range data.

4.1.2. Carve a cube at a time. Our algorithm discretizes space into a collection of cubes or voxels and processes them one at a time. Figure 13 illustrates how the status of a single cube with respect to a single view is determined:

- In case (a) the cube lies between the range data and the sensor. Therefore the cube must lie outside of the object and can be discarded.
- In case (b) the whole cube is behind the range data. As far as the sensor is concerned, the cube is assumed to lie inside of the object.
- In case (c) the cube is neither fully in front of the data nor behind the data, in most cases because it intersects the range map. The cube is assumed to intersect the object surface.

The cubes are labeled as follows. The eight corners of the cube are projected to the sensor's image plane, where their convex hull forms a hexagon. The rays from the sensor to the hexagon form a cone, which is truncated so that it just encloses the cube (see Fig. 14a). If all the data points projecting onto the hexagon are behind the truncated cone (i.e., are farther than the farthest corner of the cube from the sensor), the cube is outside the object. If all these points are closer than the closest cube corner, the cube is inside the object. Otherwise, we have the boundary case. Areas for which range data is missing are treated as points that are close to the sensor. If the data have been preprocessed so that parts of the range maps were labeled as background, the background points are treated as being infinitely far away from the sensor.

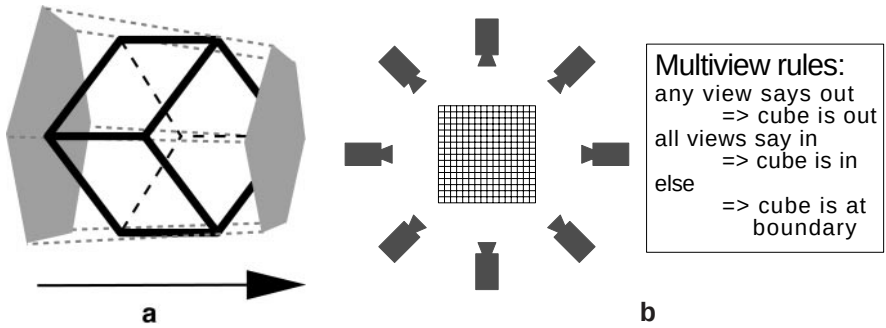


FIG. 14. (a) An octree cube and its truncated cone. The arrow points to the sensor. (b) Multiple cubes and sensors.

Figure 14b illustrates the situation in which the whole working volume has been tessellated into cubes and the scene has been scanned from several viewpoints. The cubes are processed independently from one another. Each cube is labeled with respect to all the views using the following rules. Determining that a cube is outside of the object is conclusive evidence; therefore it is enough for a single view to label a cube as outside to remove it. We can only indirectly infer that a cube is inside the object, and then only if every view agrees. If one more view were available, that view might indicate that the cube is really outside the object or on its boundary, but without that view we can only assume the cube is inside. Again, if a cube is neither in nor out, it is labeled as part of the object boundary. The implicit surface corresponding only to the observed data will then be the surface of minimum volume. The bias toward range points that are behind others is corrected during mesh optimization.

4.1.3. Hierarchical approach. A fixed partitioning of space suffers from a tradeoff between accuracy and efficiency. With large cubes the object cannot be modeled accurately. Using small cubes gives better results, but requires much more work. However, if we take a hierarchical approach using octrees, large chunks of space can be quickly ruled out and our efforts can be concentrated close to the surface.

Figure 15 illustrates the hierarchical space carving approach for the chair data set. Initially a large cube surrounds the data. Since by definition it intersects the data, it is immediately subdivided into eight smaller cubes, which the algorithm tries to label as being outside the object. In Fig. 15 none of these eight smaller cubes is carved away. At the next level a number of cubes are discarded. The remaining volume shrinks closer and closer to the actual surface until finally, in this case after seven subdivisions, we are left with a relatively accurate approximation to the chair, with the correct topological structure.

In this *simultaneous processing* order, the whole octree is traversed once, and the consensus of all the views is used to label each cube. Another possibility is *sequential processing*, where the algorithm hierarchically processes one view at a time, building on the results of the previously processed views. Sequential processing can be used to integrate a new view into a previously processed octree. The algorithm recursively descends the octree and performs the occlusion test for each cube that has not been determined to lie outside of the object. If the new view indicates that a cube is outside, the cube is relabeled and the subtrees below it are removed. Similarly, a boundary label overrides a previous inside label, in which case the cube's descendants have to be recursively tested, potentially to the maximum subdivision level.

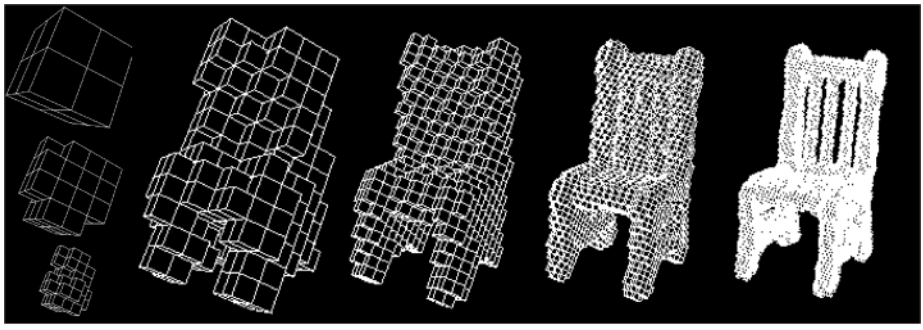


FIG. 15. Hierarchical carving of cubes.

Although both processing orders produce the same result, the simultaneous processing order is in general faster [56]. In sequential processing the silhouette of the object creates a visual cone (centered at the sensor) that separates volumes known to be outside from those speculated to be inside. The algorithm would then have to recurse to the finest subdivision level to accurately determine this boundary. In simultaneous processing, however, another view could determine at a rather coarse level of subdivision that at least part of that boundary is actually outside the object, and the finer levels of the octree for that subvolume need never be processed.

The sequential processing approach has a different kind of advantage. Since we only need to store the data from a single view at a time, this approach scales better when there are a large number of views of the same object. If we have dozens or even more views, we can use a hybrid between the sequential and simultaneous approaches. First, choose a small subset of the views (8–12) from different sides of the object. The subset is processed using the simultaneous approach, and most of the excess space is carved away. The first set of views is discarded, and a new subset is selected and processed using the current model, until all the views have been processed.

4.1.4. Mesh extraction. The labeling of the octree nodes divides the space into two sets: the cubes that are known to lie outside the object and the cubes that are assumed to be part of the object. Our surface estimate will be the closed boundary between these sets. This definition allows us to create a plausible surface even at locations where no data are obtained. The boundary is represented as a collection of vertices and triangles that can easily be combined to form a mesh.

The octree generated by the algorithm has the following structure: outside cubes and inside cubes do not have any children, while the boundary cubes have a sequence of descendants down to the finest subdivision level. To generate a surface representation, we traverse the octree starting from the root. At an outside cube we do not need to do anything. At a boundary cube that is not at the finest level, we recursively descend to the children. If we reach the maximum subdivision level and the cube is either at the boundary or inside, we check the labelings of the cube's six neighbors. If a neighboring cube is an outside cube, two triangles are created for the face they share. In an inside cube that is not at the maximum subdivision level, the algorithm checks whether it abuts an outside cube, and if so creates enough triangles (of the same size as the ones created at the finest level) to cover the shared part of the face.

4.2. *Mesh Optimization*

The hierarchical space carving algorithm produces a mesh that is not a very good approximation for the object surface. The mesh is dense, and the orientations of the mesh faces suffer from severe quantization as they are strictly aligned with the coordinate axes. However, the mesh is a proper 2D manifold and its vertices are relatively close to the object surface, so it serves as an excellent starting point for nonrobust optimization methods.

The mesh optimization method we adopted is described by Hoppe *et al.* [25, 28]. The approach tries to fit the initial mesh to a set of unorganized points, while at the same time simplifying the structure of the mesh. Clearly these two goals are conflicting: the more faces there are in the mesh, the more accurately the data can be approximated. However, a concise surface description is faster to operate on, to transfer, and to display. A practical goal is therefore a compact but still fairly accurate mesh.

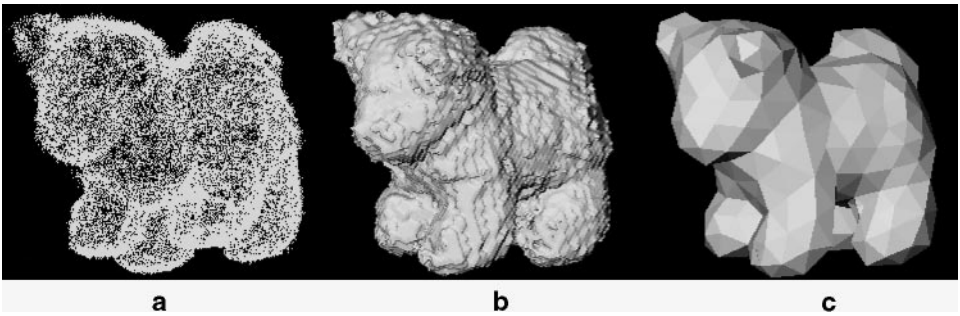


FIG. 16. (a) Point cloud of toy dog. (b) Initial smoothed mesh created by hierarchical space carving, 49,316 faces. (c) A simplified mesh, 576 faces.

Figure 16 shows a typical mesh optimization sequence using the method presented above. We start with a point cloud (Fig. 16a) and a dense initial mesh (Fig. 16b) created by our hierarchical space carving algorithm. Figures 16c–16f show four snapshots from the optimization process. The above method requires the setting of a representation penalty, which is increased monotonically during the iteration, as well as a spring constant. In this example the spring constant seems to have been too low for this noisy data at first, resulting in a roughly chiseled mesh. At later iterations higher spring constants and representation penalties yield a concise but accurate model of a toy dog.

4.3. *Discussion*

Our work is not the first to use octrees for hierarchical processing of the working volume. Szeliski [56] constructed octree bounding volumes of objects from silhouettes extracted from intensity images. Each silhouette forms a cone centered at the corresponding camera, and the resulting model is the intersection of these visual cones. In the limit (with infinite number of input images) the resulting model is called the line hull. The objects do not have to be convex for this technique to work; some holes can be recovered. However, only surface locations that are part of the silhouette from some viewpoint can be modeled accurately, so indentations and locations where both the principle curvatures are negative cannot be recovered. Connolly [10] created octree models from range maps taken from arbitrary viewpoints. Each range map is converted to a quadtree, and the quadtrees are transformed into the octree coordinate system and then assimilated into an octree model of the object. The algorithm projects four rays for each quadtree node through the octree, clearing octree nodes along the way. Since the level of octree nodes is the same as the level of the quadtree node, the hole that is carved is jaggy and larger than it should be. The carving could be made more accurate by performing it at a finer level of resolution, but then the processing would become more costly, and it would become more difficult to guarantee that all the cubes that should be removed are removed. Chien *et al.* [9] tried to address these problems by creating range quadtrees from six orthogonal viewing directions and intersecting them into a volumetric octree model. Although the quadtrees conform much better to the structure of the octree, there is a large class of objects that cannot be modeled using this approach, including objects with at least two holes that are not perpendicular to each other. Our approach of projecting subvolumes onto range images is a much more robust method; it eliminates the deficiencies of Connolly's without introducing the restrictions of Chien *et al.*'s approach. Other works that use a principle similar to ours include [33, 57]. A recent paper by Kutulakos

TABLE 1
Statistics for the Chair Data Set

Levels:	2	3	4	5	6	7	8
Nodes:	73	217	649	2377	9393	41201	190441
Faces:	74	128	422	1372	5152	18446	71038
Seconds:	0.1	1.2	0.8	1.7	6.0	26.1	121.1

Note. For example, at level 5 the octree contained 2377 nodes, the boundary between the object and free space was 1372 faces, and it took an additional 1.7 s to process from level 4.

and Seitz [31] formally studies space carving and combines basic shape-from-silhouettes [56] with ideas from shape-from-stereo and shape-from-shading. Their method does not require explicit range data, requiring instead a set of calibrated color images with known poses and that the scene radiance follow a locally computable model such as Lambertian reflection.

In the rest of this section we discuss some execution statistics, the connection between our approach and interval analysis, how our method deals with thin objects, and how it is affected by noisy input data.

4.3.1. Execution statistics. Table 1 summarizes some execution statistics for the chair data set. It shows the number of octree nodes, the number of square faces between the outside cubes and the others, and execution times for subdivision levels 2 through 8. The timings were obtained using an SGI O2 with a 175-MHz R10000 processor, and they show the incremental extra time it takes to process a level given that all the data were already processed up to the previous level. From these results we can make several observations. First, the execution times are fast enough to allow interactive selection of a suitable processing resolution, balancing the competing goals of correct recovery of object topology and conciseness of the representation. Second, the size of internal data structures, output, and execution times all grow exponentially as powers of (approximately) four. The theoretical reason for this is quite obvious. Each increase in level causes the octree cells to be bisected in three orthogonal directions, so the number of cells grows by a factor of $2^3 = 8$. However, the object surface is a 2D manifold embedded in 3D, so the number of cells that intersect the surface grows only by a factor of $2^2 = 4$. The fact that time and space requirements grow approximately by factors of four demonstrates that the algorithm automatically concentrates its efforts close to the surface. There is a seeming anomaly in the timing for level 3. The probable reason is that at the coarse levels the cubes' projections onto the sensors' image planes cover a large area, so the inside/outside labeling of the cubes takes a long time.

4.3.2. Interval analysis. Our guideline in surface reconstruction was to avoid nonrobust techniques such as averaging as much as possible. Instead, we propagate and manipulate constraints into more useful forms that allow robust estimation of the object surface topology, while we leave the accurate fitting of a concise surface representation to the data to a later stage. The space carving method that we use to robustly determine surface topology is a particular instance of a general technique called interval analysis.

Interval analysis [53] takes a conservative approach for determining function values. Let us assume that we want to plot a function $f(x)$, and we do not know the actual values, but we can somehow deduce the minimum and maximum values f attains within a given interval. Rather than plotting some approximation of f, we can plot a block covering all the possible

values within the bounding box. Or, if we can obtain better bounds for f within a shorter interval, we can subdivide the original interval into shorter segments. Thus, interval analysis allows us to narrow down the interval within which the function value lies by eliminating the impossible intervals, ideally converging arbitrarily close to the actual value.

In our reconstruction algorithm we use interval analysis to determine binary function values that indicate whether a point in a volume is inside or outside an object. We partition space into regions (cubes) and conservatively determine for each region whether it lies completely inside, completely outside, or neither. Using recursive subdivision, we efficiently prune large regions of space away from the object boundary, and focus both computation and storage on the spatial region of interest, that is, the region next to the surface.

4.3.3. Thin objects. Some methods that employ signed surface distance functions are unable to correctly reconstruct thin objects. Curless and Levoy [12], for example, build a distance function by storing positive distances to voxels in front of the surface and negative distances to voxels behind the surface. In addition to distances, weights are stored to facilitate combining data from different views. With this method, views from opposite sides of a thin object interfere and may cancel each other, preventing reliable extraction of the zero set of the signed distance function.

In the case of a thin sheet of paper, our algorithm would construct a thin layer of octree cubes (voxels) straddling the paper by carving away the space between the sensor and the object but leaving the voxels that intersect the object, creating a shell around the surface. This is illustrated in Fig. 17a. Note, however, that our method can fail in the presence of measurement noise and registration error if the minimum cube size is set too small, as illustrated in Fig. 17b. Due to a registration error the indentation on the light gray view penetrates in front of the darker view, and vice versa. This causes too many cubes to be carved away, creating a hole. This example shows that in order to correctly reconstruct thin objects, the minimum size for the cubes should be larger than the sum of the registration error, the range sampling error, and the sampling density.

4.3.4. Effects of noise. One of the benefits of the space carving approach is that it automatically removes most of the outliers in the input data. Suppose that our first view contains some outlier data points that fall outside the object. Because we have only a single view, our algorithm first labels the cubes containing the outliers as lying on the object surface. However, if from the perspective of another view one of these cubes lies in front of the object (or even background), the cube is removed, thereby eliminating the corresponding outlying data. This may also happen if the outlying point is a valid data

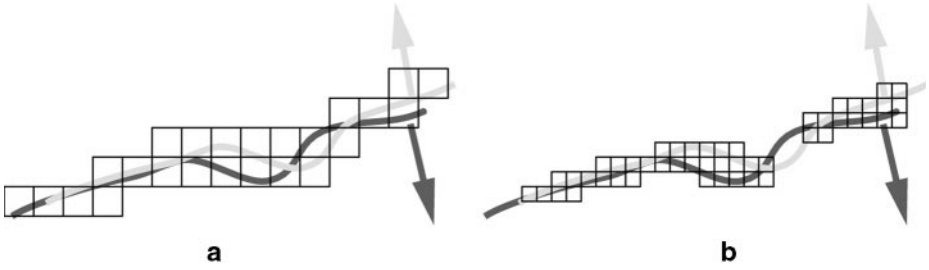


FIG. 17. A thin sheet seen from the left (light gray) and right (dark gray) is reconstructed correctly in (a). In (b) registration error and small cube size combine to cause a hole.

point from the background instead of an erroneous range measurement. Suppose that the views were obtained by reorienting the object in front of a stationary scanner and that the views have been registered into a common coordinate system using a subset of the surface points. Now the background moves relative to the object from view to view, and in general the background data in one view can be identified by another view and labeled as such or removed altogether.

Outliers that lie behind the surface as seen from the scanner can be damaging, since they could cause our algorithm to incorrectly carve away a piece of the object. For several reasons, this has not been a problem for us. Most outliers we have observed either belong to other objects in the background or appear around the silhouette edges of the object. We use background data to carve the space around the object more efficiently, and outliers at the object boundaries do not cause deep holes to be carved into the object.

Noise still remains a problem for the mesh optimization phase. However, using the initial mesh we can first eliminate input points that are too far from the mesh, as they are likely to be outliers.

5. VIEW-DEPENDENT TEXTURING

Our scanner obtains color images along with range data, and in Section 3 we described how we use the color information to aid in obtaining correct 3D registration. Our surface reconstruction method did not use color information, but we want to be able to associate the input color with the resulting surface models. Although the coarse polygon mesh in Color Plate 2 (left) is recognizable as a toy dog, the textured model in Color Plate 2 (right) almost completely hides the polygonal nature of the underlying surface representation, and the intricate fur texture gives an impression of very detailed geometry.

Instead of statically associating a single color with each surface location, we use *view-dependent* texturing to dynamically project color onto the surface depending on the current viewpoint. View-dependent texturing was initially used in *image-based rendering* systems [6, 22, 32, 41], some of which do not use any geometric information, but it is also increasingly being used to texture map geometric models [15, 39, 42, 47].

View-dependent texturing using several existing color images is really an interpolation problem. When rendering an image of the surface model with colors, we think in terms of (inverse) rays of light that propagate through pixels on the image plane and intersect the surface model. When determining the color for the ray–surface intersection, we first have to identify the images that potentially may see that location. We then must identify the pixels within those images that we need. Finally, input color values from several images have to be blended together. The following three subsections describe how we solve these problems in order to determine the color of a given pixel in the output image. We then explain how we can obtain real-time renderings using these techniques, present some results, and conclude with a discussion. A more detailed description comparing this and an alternative view-based rendering method can be found in [47].

5.1. *Choosing Views*

In principle, any camera view that sees the same surface point as visible through a viewer pixel could contribute to the color of that pixel. However, views with viewing directions far away from that of the virtual camera should not be used if closer views are available. Otherwise, self-occlusions become much more frequent, so only a small portion of the surface

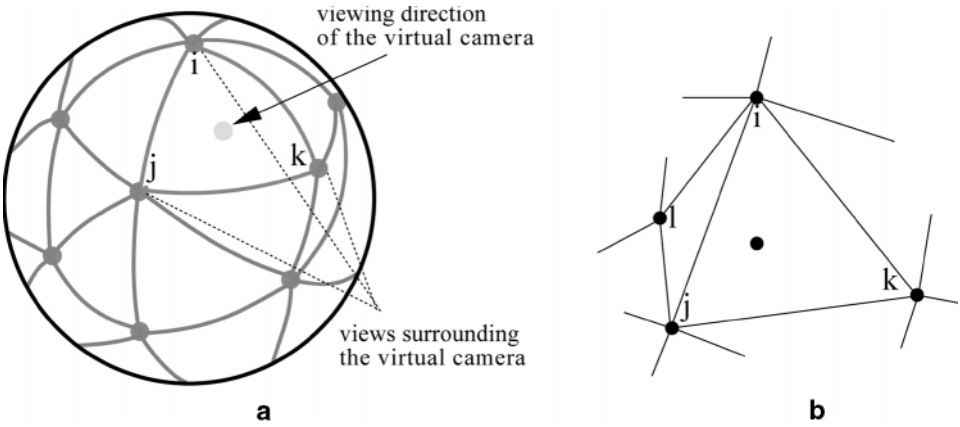


FIG. 18. (a) A Delaunay triangulation of the unit sphere is computed for the vertices corresponding to the viewing directions of the input images. The triangle (i, j, k) containing the viewing direction of the virtual camera determines the three views used to search for candidate rays. (b) Although view l is closer to the current viewing direction, view k is a better choice.

is likely to be visible both to the viewer and to the distant view, if any. Additionally, small errors in registration, camera calibration, and surface reconstruction lead to larger errors in backprojecting surface points to the color images. In our system we only search for matching pixels from three input images that have been taken from nearby viewing directions.

To facilitate the selection of suitable views, the views are organized as illustrated in Fig. 18a. Vertices corresponding to each of the viewing directions of the input images are placed on a unit sphere, and a Delaunay triangulation of the sphere using those vertices is computed. To begin rendering a frame, we determine the viewing direction of the viewer and place a corresponding vertex on the unit sphere as shown in Fig. 18a. The triangle containing that vertex determines the three views within which candidate rays for the surface points visible to the viewer will be sought. Note that these views are not always the three closest views, although the closest one is guaranteed to be among the three. For example, in Fig. 18b view l is closer to the current view direction than view k. However, view k is preferred because i, j, and l all lie to one side of the current view. If there is some part of the surface visible to the viewer but occluded in views i and j, that location is more likely to be visible in view k than in view l.

5.2. *Finding Compatible Rays*

The first task in determining the color of a particular pixel in the viewer image is to locate the point on the object surface that is visible through that pixel. Color Plate 3a illustrates this problem by showing a ray through one of the viewer's pixels ending at its first intersection with the object surface.

There are two straightforward ways in which graphics hardware can be used to determine the surface point visible through a given pixel. The method we have chosen to implement (and the one taken by Gortler *et al.* [22]) is illustrated in Color Plate 3b. First, we calculate the axis-aligned bounding box for the triangle mesh representing the object. We then scale and translate the coordinates of each vertex so that the bounding box is transformed to a cube with unit-length sides. Now we can encode the x, y, and z coordinates of each vertex in the red, green, and blue components of its color, so that when the mesh is rendered in the

viewer, we get an image like the one in Color Plate 3b. Within each triangle, the graphics hardware interpolates the color, and therefore also the encoded surface location. The surface location visible through a pixel is then given by the pixel's color.

The method described above can only be applied directly to polygonal surface representations. For a more general technique, we could read in the depth buffer after rendering the untextured surface, and convert each depth value to the corresponding 3D point using the known projection and viewing transformations. However, this approach requires over four times as many basic operations (multiplications, divisions, and additions).

Once we have determined the surface point corresponding to a viewer pixel, we obtain candidate rays by projecting that point back to the input images. For example, the viewer pixel marked by the arrow in Color Plate 3b corresponds to a point on the dog's snout, which projects back to the dots in the three images in Color Plate 3c. To perform these projections, we need to know each camera's internal and external parameters. In our case, the camera calibration procedure used for obtaining depth from stereo determines the internal parameters of the camera; the external parameters for each view are obtained by registering the range maps into a common coordinate system.

Not all the candidate rays obtained through backprojection should be accepted. Due to self-occlusion the surface point may not be visible in a given image. We can detect this if we retain the original range maps for each camera. For example, we can calculate the distance from the surface point to the camera and compare it to the range map value for the pixel to which the point projects. If these distances differ significantly, we reject the ray.

5.3. *Combining Rays*

The colors of the compatible rays are averaged together via a weighting scheme that uses three different weights: *directional* weight, *sampling quality* weight, and *feathering* weight.

The task of the directional weight is to favor rays originating from views whose viewing direction is close to that of the virtual camera. Not only should a view's weight increase as the current viewing direction moves closer, but the other views' weights should decrease, leaving only the closest view when the viewpoints coincide. Because we use three input views forming a triangle containing the current view, it is natural to choose as weights the *barycentric coordinates* of the current view with respect to the others. We radially project the vertex of the current view direction onto the planar triangle formed by the surrounding three views (i.e., the triangle is intersected by a line passing through the point and the center of the sphere). The directional weight of each of the three input views is given by the corresponding barycentric coordinate of the projected point.

The sampling quality weight directly reflects how well a ray samples the surface. We assign to each ray/pixel of each input image a weight that is defined as the cosine of the angle between the local surface normal and the direction from the surface point to the sensor. This weight is illustrated in Fig. 19b for a view of the toy dog (the higher the intensity, the higher the weight).

The feathering weight is used to hide artifacts due to differences in lighting among the input images and other view-dependent differences. Without the feathering weight, the boundaries of the input views become noticeable because a view contributes to pixel colors on one side of a silhouette edge but not on the other. As illustrated in Fig. 19c, the feathering weight is zero outside the object, and it grows linearly to a maximum value of one within the object.

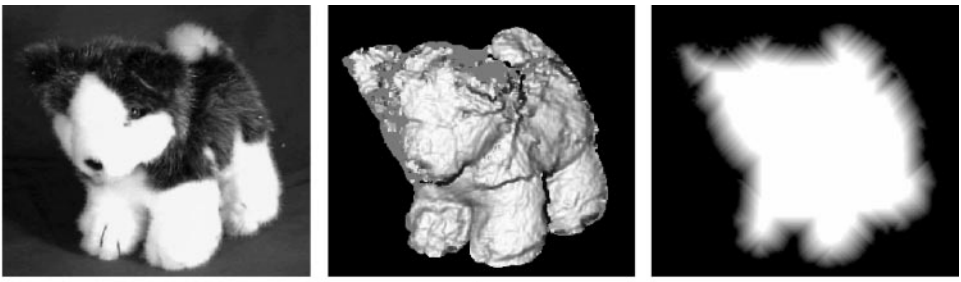


FIG. 19. (a) A view of a toy dog. (b) The sampling quality weight. (c) The feathering weight.

5.4. *Precomputation for Runtime Efficiency*

In principle, the directional weight changes for each pixel. However, in order to accelerate the implementation, we choose three input images for all the pixels of a given frame using the general viewing direction (the direction of the optical axis) as the direction of the ray. Similarly, the directional weight of each input image is calculated once for a frame and is the same for every pixel. Of course, in a 3D viewer, if the viewpoint changes, the input images and directional weights are recalculated.

On the other hand, the sampling quality and feathering weights depend only on the input data and are not functions of viewpoint. Therefore they remain constant, so we can determine them in advance and store them in a lookup table. We preprocess each pixel of each image by calculating these two weights and storing their product in the alpha channel of a pixel, where it is readily accessible.

A large part of the viewer's processing time is spent projecting object surface points onto input images. We speed this up by preprocessing the input images (along with associated information such as the weights and range data) to remove the radial lens distortions beforehand, so that only perspective projection remains for runtime.

5.5. *Results*

Our approach does not require hardware texture mapping, yet we can display complex textured models at interactive rates (5 to 6 frames per second on a 175-MHz SGI O2). The only part of the algorithm that uses hardware graphics acceleration is the rendering of z-buffered Gouraud-shaded polygons to determine which points on the object surface are visible in each frame. The algorithm can be easily modified to work with arbitrary surface descriptions by finding the visible surface points using the z-buffer instead of rendering triangles colored by location.

5.6. *Discussion*

5.6.1. Related work. The following four works inspired us to think about view-based texturing in terms of finding and blending rays. Chen [5] and McMillan and Bishop [41] modeled environments by storing the light field function around a point. The rays visible from a point are mapped to a cylinder around that point, and new horizontal views are created by warping a portion of the cylinder to the image plane. Both systems allow rotations about the vertical axis, but they do not support continuous translation of the viewpoint. Levoy

and Hanrahan [32] and Gortler *et al.* [22] developed image synthesis systems that use a 4D slice of the light field and that support continuous translation and rotation of the viewpoint. In both systems, new images are synthesized from a stored grid of 2D images by an interpolation procedure, but Gortler *et al.* use additional geometric information to improve on ray interpolation. One advantage of these methods over surface reconstruction techniques is that they can capture the appearance of any object regardless of the complexity of its surface. The main disadvantage is the difficulty of storing and accessing its enormous representation.

Debevec *et al.* [14, 15] developed a system that fits user-generated geometric models (blocks, arcs, etc.) of buildings to digitized images by interactively associating image features with model features and adjusting model parameters to fit the images. View-dependent textures are applied to the buildings by projecting the color images onto the geometric model. If more than one image projects to the same location, images taken from view directions close to that of the virtual camera are weighted more heavily, but the exact weighting function is not described. A blending weight is used close to the boundaries in the input data; our feathering weight performs the same function. Although Debevec *et al.* mention the importance of taking image resolution or sampling quality into account, they leave that for future work.

There are a number of articles related to our work that focus not on the display of scanned objects, but rather on the rapid rendering of complex environments [6, 13, 36, 40, 52]. The idea is to trade unbounded scene complexity for bounded image complexity.

5.6.2. Geometry vs sampling density. One of the key motivations for our approach was to study how one can trade off geometry and color sampling density. A pure image-based approach such as light field rendering [32] requires a very dense sampling of the light field in order to provide realistic images from arbitrary view directions. The sparser the sampling of the color images, the more blurring there is, as the aperture of the camera has to be increased to avoid aliasing artifacts. For most surfaces the light field function changes smoothly directionally, which may enable the light fields to be drastically compressed. First, we would argue that knowing even approximate geometry can make this compression easier and perhaps more efficient. Second, if one can acquire both color and range data, view-dependent texturing can provide this compressed form of light fields directly from a sparse input set.

6. SUMMARY AND FUTURE WORK

In this paper, we have presented a complete system that begins with scanning both the geometry and color of real objects and finally displays realistic images of those objects from arbitrary viewpoints at interactive speeds. In the following subsections we summarize our data acquisition system and the methods for registration, surface reconstruction, and view-dependent texturing. After each summary, we discuss possible related future work.

6.1. *Data Acquisition*

We have built an active stereo camera system that scans both the geometry and the color of an object. Our system solves the correspondence problem by the use of active lighting; i.e., we project a moving vertical light stripe onto the surface. From corresponding image locations we calculate 3D point samples on the object surface.

Traditional methods for locating the center of the light stripe bias result in discontinuities of the surface, its orientation, and its color. For scanners that sweep the light stripe at a constant rate across an object, it is possible to accurately track the motion of the stripe and infer location more reliably from the estimated motion. The incorporation of spacetime analysis [11] significantly improved the quality of our range data.

6.1.1. Future work: continuous surface digitization.[1] Rather than discretely finding for each pixel the corresponding pixel on the other images, even if that could be done in subpixel accuracy, one could try to fit a piecewise continuous surface to the timespace data of each camera. This surface is a height field over the image, where the "height" is the time. The center of the beam in the ith image would then be the piecewise continuous intersection curve of that surface and a plane that is the constant i over the whole image. One could then imagine taking such curves from two cameras, extending the curves in the camera images into 3D surfaces using the camera calibration data, and intersecting the 3D surfaces. The intersection curve lies on the object surface. This way, instead of digitizing discrete surface points, we could digitize piecewise continuous curves on the surface. We could further extend this approach to digitize piecewise continuous surfaces: project the isocontour curves into 3D *continuously* in time, intersect the projection surfaces, and create a piecewise continuous surface in 3D parameterized by the image rows of one of the cameras and the time.

6.2. *Registration*

Most registration methods assume an approximate initial registration, which is then iteratively refined until the process converges. Typically scanned points in one view are paired with points in another view, using various heuristics so that the paired points correspond to the same surface points. With those pairs the surfaces can be moved closer together, and the hope is that the point-pairing becomes easier and more reliable. We have developed a better point-pairing method that simultaneously establishes all the correspondences between the two scans by projecting the colored geometry onto the scanner image plane, moving one of the scans until the color images align, and pairing surface points projecting to the same point on the image plane. The range data are then moved closer using a robust statistical regression method. This approach pairs most points with their true corresponding points in the other view and typically reaches the final accuracy in a single iteration, while traditional methods take much longer and may even produce inferior results.

When a large collection of views are simultaneously registered, the correspondence problem becomes even more severe. Our solution begins with registering each view with several neighboring views in a pairwise fashion. After a view has been registered with respect to another view, it is easy to find corresponding points simply using spatial proximity. The established pairings are stored, and finally the registration transformations for all the views are found simultaneously using the stored point pairs.

6.2.1. Future work: extend the image alignment phase. Our current method first aligns the color data of two scans so that a reliable point correspondence can be determined, followed by geometry alignment using those correspondences. Instead of performing the registration in two steps, we could use the range data of both the views to move one of the

[1] This idea came about in a discussion with Brian Curless.

meshes so that both the color data and the height fields are aligned, along with any other functions associated with the surface (such as normal vectors). This way we could minimize at the same time a weighted combination of color and geometry rather than iterate between the two subproblems.

6.3. *Surface Reconstruction*

The most difficult part of surface reconstruction is to correctly determine the connectivity of the surface. If we are first given a rough estimate of the surface with fixed topology, the problem becomes a much easier function estimation problem. We developed a hierarchical space carving algorithm that is robust in the case of missing and noisy data. The algorithm partitions space into an octree and hierarchically labels it to be inside, outside, or on the object boundary. Finally, it extracts the boundary between the outside cubes and all the other cubes. The result is an approximate surface close to the actual surface. The surface is a proper 2D manifold; possible holes due to missing data are filled with a plausible surface.

The initial mesh that we construct using our hierarchical space carving algorithm is a starting point to obtain a mesh that more closely fits the data and is as sparse as possible, while still being able to closely approximate the object. We use the Hoppe *et al.* [28] algorithm for the mesh optimization. The method iterates between modifying mesh vertex locations to better fit the mesh to the data and stochastically simplifying the mesh.

6.3.1. Future work: increase geometric accuracy. Our space carving method produces a blocky surface, as each face can only be parallel or perpendicular to the neighboring faces. We could post-process the resulting mesh to better conform to the data by projecting the mesh vertices into the data. We could in fact use the same weighting scheme that Curless and Levoy used. The difference is that they first estimated the surface distance function and then extracted the surface. In our case the mesh is extracted first, and now we can better deal with thin surfaces: all we have to do is to make sure that the vertices are not positioned so that the surface self-intersects.

6.4. *View-Dependent Texturing*

The visual appearance of an object is determined not only by its geometry, but also by the surface color and texture. The less uniform the color, the more important role it typically plays. We have two reasons to use view-dependent texturing, where the texture used to determine the color changes depending on the viewing direction, rather than static texturing, where each surface element has exactly the same colors independent of the viewing direction. The first reason is that the scanned data is view dependent: the accuracy and sampling density of both geometric and color data depend on the relative orientations of the surface and scanning direction. The second and more important reason is that we can use view-dependent texturing to compress geometry. The original surface may contain fine detailed geometry and thus appears different when viewed from different directions. Even if our approximation does not directly capture that fine geometry, we can capture its appearance using view-dependent texturing.

Our method for view-dependent texturing of reconstructed surfaces using color images in essence projects a few color images onto an approximate surface model. In practice, the projection starts from the virtual viewer where a ray is projected from the viewpoint

through a pixel. The ray is intersected with the surface model, and the intersection point is projected back to the color images, producing a candidate color from each image for the original pixel in the viewer. The candidate colors are blended using weights that depend on how closely the viewing direction of a color image matches that of the virtual viewer, how well the candidate pixel samples the surface, and how close the candidate pixel is to the object silhouette. The method integrates the geometric data in 3D and then projects color data onto the image using the geometric model.

6.4.1. Future work: relighting. Our view-based texturing approach can only be used to view objects in the lighting conditions that prevailed when the color images were obtained. It is possible to extend our method to allow relighting of the object in different illumination conditions. The idea is to store a reflectance distribution function in place of a color for each pixel of each input image.

A practical setup would require arranging several light sources at various known positions around the working volume of the scanner. First, the object geometry is scanned once. Then, without moving the object or the cameras, several color images are taken, each under different known lighting conditions. From this information, we can estimate a unidirectional reflectance distribution function for each pixel. The distribution gives the observed radiance as a function of wavelength and the direction of the incoming light. Some results using synthetic models and lighting are reported by Wong *et al.* [62].

In our current method, the color value of a contributing image ray is determined by looking up a stored RGB value. The only difference in the proposed method would be to change that lookup to an evaluation of the reflectance function associated with the ray. Notice that this approach would automatically encode many lighting effects, such as self-shadowing, diffuse interobject reflections, and to a certain degree, even specular highlights, without the need to model them explicitly.

APPENDIX: LIST OF SYMBOLS

$\mathbf{T}$	rigid transformation (rotation and translation)
$\mathbf{p}$, $\mathbf{q}$	scanned points (or surface points)
$\mathbf{P}$, $\mathbf{Q}$	sets of scanned points (or surfaces)
(u, v)	image coordinates
ψ	image flow
$\mathbf{w}$	weight
I_0, I_1	color images
$\mathbf{g}$	color gradient
$\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$	unit coordinate vectors
$\mathbf{x}_0$, $\mathbf{x}_1$	origin of a coordinate system
$\mathbf{f}$	focal length
$\mathbf{c}$	center of mass (and of rotation)
α, β, γ	rotation angles
$\mathbf{b}$	translation vector
$\mathbf{h}$	height field
r	residual
ϵ	rate of outliers
s^0	estimate for standard deviation

ACKNOWLEDGMENTS

This research was conducted at the University of Washington in Seattle in 1994–1997, and it was supported by grants from NSF (IRI-9520434 and DMS-9402734), Academy of Finland, Emil Aaltonen Foundation, Finnish Culture Foundation, and Human Interface Technology Laboratory. We also thank Habib Abi-Rached, Michael Cohen, Tony DeRose, Tom Duchamp, Hugues Hoppe, John McDonald, and Werner Stuetzle for their input at various phases of this work.

REFERENCES

1. P. J. Besl, *Surfaces in Range Image Understanding*, Springer-Verlag, Berlin/New York, 1988.
2. P. J. Besl and N. D. McKay, A method for registration of 3-d shapes, *IEEE Trans. Patt. Anal. Machine Intell.* **14**(2), 1992, 239–256.
3. G. Blais and M. D. Levine, Registering multiview range data to create 3d computer objects, *IEEE Trans. Patt. Anal. Machine Intell.* **17**(8), 1995, 820–824.
4. G. Champleboux, S. Lavallée, R. Szeliski, and L. Brunie, From accurate range imaging sensor calibration to accurate model-based 3-d object localization, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1992*, pp. 83–89.
5. S. E. Chen, Quicktime VR—An image-based approach to virtual environment navigation, In *SIGGRAPH 95 Conference Proceedings*, pp. 29–38, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1995.
6. S. E. Chen and L. Williams, View interpolation for image synthesis, in *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 279–288.
7. Y. Chen, *Description of Complex Objects Using Multiple Range Images*, Ph.D. thesis, Institute for Robotics and Intelligent Systems, University of Southern California, 1994. Also technical report IRIS-94-328.
8. Y. Chen and G. Medioni, Object modelling by registration of multiple range images, *Image Vision Comput.* **10**(3), 1992, 145–155.
9. C. H. Chien, Y. B. Sim, and J. K. Aggarwal, Generation of volume/surface octree from range data, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '88)*, pp. 254–260.
10. C. I. Connolly, Cumulative generation of octree models from range data, in *Proc. IEEE Int. Conf. on Robotics and Automation, 1984*, pp. 25–32.
11. B. Curless and M. Levoy, Better optical triangulation through spacetime analysis, in *Proc. IEEE Int. Conf. on Computer Vision (ICCV), 1995*, pp. 987–994.
12. B. Curless and M. Levoy, A volumetric method for building complex models from range images, in *SIGGRAPH 96 Conference Proceedings*, pp. 303–312, ACM SIGGRAPH. Addison–Wesley, Reading, MA, 1996.
13. L. Darsa, B. C. Silva, and A. Varshney, Navigating static environments using image-space simplification and morphing, in *Proc. 1997 Symposium on Interactive 3D graphics.*
14. P. Debevec, *Modeling and Rendering Architecture from Photographs*, Ph.D. thesis, Department of Computer Science, University of California at Berkeley, 1996.
15. P. E. Debevec, C. J. Taylor, and J. Malik, Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach, in *SIGGRAPH 96 Conference Proceedings*, pp. 11–20, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1996.
16. C. Dorai, G. Wang, A. K. Jain, and C. Mercer, From images to models: Automatic 3D object model construction from multiple views, in *Proceedings of the 13th IAPR International Conference on Pattern Recognition, 1996*, pp. 770–774.
17. D. W. Eggert, A. W. Fitzgibbon, and R. B. Fisher, Simultaneous registration of multiple range views for use in reverse engineering, in *Proceedings of the 13th IAPR International Conference on Pattern Recognition, 1996*, pp. 243–247.
18. O. D. Faugeras and M. Hebert, The representation, recognition, and locating of 3-d objects, *Int. J. Robotic Res.* **5**(3), 1986, 27–52.
19. H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau, Registration of multiple range views for automatic 3-d model building, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1994*, pp. 581–586.

20. M. Garland and P. S. Heckbert, Surface simplification using quadric error metrics, in *SIGGRAPH 97 Conference Proceedings*, pp. 209–216, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1997.
21. G. Godin, M. Rioux, and R. Baribeau, Three-dimensional registration using range and intensity information, in *Videometrics III*, pp. 279–290, Proc. SPIE, Vol. 2350, SPIE—Int. Soc. Opt. Eng., Bellingham, WA, 1994.
22. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, The lumigraph, in *SIGGRAPH 96 Conference Proceedings*, pp. 43–54, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1996.
23. K. Higuchi, H. Delingette, M. Hebert, and K. Ikeuchi, Merging multiple views using a spherical representation, in *Proceedings of the 2nd CAD-Based Vision Workshop, 1994*, pp. 124–131.
24. A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt, Reliable surface reconstruction from multiple range images, in *Computer Vision—ECCV '96*, Springer, Berlin/New York, 1996.
25. H. Hoppe, *Surface Reconstruction from Unorganized Points*, Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, June 1994.
26. H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, Piecewise smooth surface reconstruction, in *Proceedings of SIGGRAPH '94*.
27. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, Surface reconstruction from unorganized points, in *Proceedings of SIGGRAPH '92* pp. 71–78.
28. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, Mesh optimization, in *Computer Graphics (SIGGRAPH '93 Proceedings)*, Vol. 27, pp. 19–26.
29. B. K. P. Horn, Closed-form solution of absolute orientation using unit quaternions, *J. Opt. Soc. Am. A* **4**(4), 1987, 629–642.
30. H. Jin, T. Duchamp, H. Hoppe, J. A. McDonald, K. Pulli, and W. Stuetzle, Surface reconstruction from misregistered data, in *Vision Geometry IV*, pp. 324–328, Proc. SPIE, Vol. 2573, SPIE—Int. Soc. Opt. Eng., Bellingham, WA, 1995.
31. K. N. Kutulakos and S. M. Seitz, A theory of shape by space carving, in *Proc. IEEE Int. Conf. on Computer Vision (ICCV), 1999.*
32. M. Levoy and P. Hanrahan, Light field rendering, in *SIGGRAPH 96 Conference Proceedings*, pp. 31–42, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1996.
33. A. Li and G. Crebbin, Octree encoding of objects from range images, *Pattern Recognition* **27**(5), 1994, 727–739.
34. C. Loop, *Smooth Subdivision Surfaces Based on Triangles*, Master's thesis, Department of Mathematics, University of Utah, 1987.
35. W. E. Lorensen and H. E. Cline, Marching cubes: A high resolution 3D surface construction algorithm, in *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21. pp. 163–169.
36. W. R. Mark, L. McMillan, and G. Bishop, Post-rendering 3d warping, in *Proc. 1997 Symposium on Interactive 3D graphics.*
37. T. Masuda, K. Sakaue, and N. Yokoya, Registration and integration of multiple range images for 3-D model constuction, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1996*, pp. 879–883.
38. T. Masuda and N. Yokoya, A robust method for registration and segmentation of multiple range images, *Comput. Vision Image Understanding* **61**(3), 1995, 295–307.
39. Y. Matsumoto, H. Terasaki, K. Sugimoto, and T. Arakawa, A portable three-dimensional digitizer, in *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling, 1997*, pp. 197–204.
40. N. Max and K. Ohsaki, Rendering trees from precomputed Z-buffer views, in *Eurographics Rendering Workshop 1995*, pp. 74–81, 359–360.
41. L. McMillan and G. Bishop, Plenoptic modeling: An image-based rendering system, in *SIGGRAPH 95 Conference Proceedings*, pp. 39–46, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1995.
42. W. Niem and J. Wingbermühle, Automatic reconstruction of 3d objects using a mobile monoscopic camera, in *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling, 1997*, pp. 173–180.
43. R. Pito, Mesh integration based on co-measurements, in *Proc. IEEE Int. Conf. on Image Processing, Special Session on Range Image Analysis, 1996.*
44. M. Potmesil, Generating models for solid objects by matching 3d surface segments, in *Proc. Int. Joint Conf. on Artificial Intelligence, 1983*, pp. 1089–1093.

45. K. Pulli, *Surface Reconstruction and Display from Range and Color Data*, Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, 1997.
46. K. Pulli, H. Abi-Rached, T. Duchamp, L. G. Shapiro, and W. Stuetzle, Acquisition and visualization of colored 3d objects, in *Proc. Int. Conf. on Pattern Recognition, 1998.*
47. K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle, View-based rendering: Visualizing real objects from scanned range and color data, in *Proc. 8th Eurographics Workshop on Rendering, 1997.*
48. K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle, Robust meshes from multiple range maps, in *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling, 1997*, pp. 205–211.
49. P. Rousseeuw and A. Leroy, *Robust Regression and Outlier Detection*, Wiley, New York, 1987.
50. P. Rousseeuw and B. van Zomeren, Unmasking multivariate outliers and leverage points, *J. Am. Statist. Assoc.* **85**(411), 1990, 633–651.
51. M. Rutishauser, M. Stricker, and M. Trobina, Merging range images of arbitrarily shaped objects, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1994*, pp. 573–580.
52. J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder, Hierarchical image caching for accelerated walkthroughs of complex environments, in *SIGGRAPH 96 Conference Proceedings*, pp. 75–82, ACM SIGGRAPH, Addison–Wesley, Reading, MA, 1996.
53. J. Snyder, Interval analysis for computer graphics, in *Proceedings of SIGGRAPH '92*, pp. 121–130.
54. M. Soucy and D. Laurendeau, A dynamic integration algorithm to model surfaces from multiple range views, *Mach. Vision Appl.* **8**(1), 1995, 53–62.
55. A. J. Stoddart and A. Hilton, Registration of multiple point sets, in *Proceedings of the 13th IAPR International Conference on Pattern Recognition, 1996*, pp. 40–44.
56. R. Szeliski, Rapid octree construction from image sequences, *CVGIP: Image Understanding* **58**(1), 1993, 23–32.
57. G. H. Tarbox and S. N. Gottschlich, IVIS: An integrated volumetric inspection system, in *Proceedings of the 1994 Second CAD-Based Vision Workshop*, pp. 220–227.
58. R. Y. Tsai, A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses, *IEEE J. Robotics Automation* **RA-3**(4), 1987, 323–344.
59. G. Turk and M. Levoy, Zippered polygon meshes from range images, in *Proceedings of SIGGRAPH '94*, pp. 311–318.
60. Z. Wang and A. Jepson, A new closed-form solution for absolute orientation, in *Proc. Conf. on Computer Vision and Pattern Recognition, 1994*, pp. 129–134.
61. S. Weik, Registration of 3-d partial surface models using luminance and depth information, in *Proc. Int. Conf. on Recent Advances in 3-D Digital Imaging and Modeling, 1997*, pp. 93–100.
62. T.-T. Wong, P.-A. Heng, S.-H. Or, and W.-Y. Ng, Image-based rendering with controllable illumination, in *Proc. 8th Eurographics Workshop on Rendering, 1997.*
63. Z. Zhang, Iterative point matching for registration of free-form curves and surfaces, *Int. J. Comput. Vision* **13**(2), 1994, 119–152.