Extracting Objects from Range and Radiance Images

Yizhou Yu, Andras Ferencz, *Student Member*, *IEEE*, and Jitendra Malik

Abstract—In this paper, we present a pipeline and several key techniques necessary for editing a real scene captured with both cameras and laser range scanners. We develop automatic algorithms to segment the geometry from range images into distinct surfaces, register texture from radiance images with the geometry, and synthesize compact high-quality texture maps. The result is an object-level representation of the scene which can be rendered with modifications to structure via traditional rendering methods. The segmentation algorithm for geometry operates directly on the point cloud from multiple registered 3D range images instead of a reconstructed mesh. It is a top-down algorithm which recursively partitions a point set into two subsets using a pairwise similarity measure. The result is a binary tree with individual surfaces as leaves. Our image registration technique performs a very efficient search to automatically find the camera poses for arbitrary position and orientation relative to the geometry. Thus, we can take photographs from any location without precalibration between the scanner and the camera. The algorithms have been applied to large-scale real data. We demonstrate our ability to edit a captured scene by moving, inserting, and deleting objects.

Index Terms—Scene editing, object-level representation, range image segmentation, image registration, texture-mapping, image-based modeling, image-based rendering, augmented reality.

1 Introduction

Capturing real environments to faithfully recreate them on a computer screen has become an important research area. Most of the work in this field, image-based modeling and rendering [27], [6], [32], [47], [28], [18], [11], [41], [40], [42], [45], [51] has focused on static environments that can be viewed from novel viewpoints as well as under novel lighting conditions. However, challenges remain in making modifications to geometric properties, such as the relative position, orientation, and size of objects, and photometric properties, such as color or specularity. For example, we would like to animate the objects in the environment or move a statue to a different place in a virtualized museum.

An object is made up of a collection of surfaces which in turn have geometric properties such as size and shape as well as photometric properties such as color and texture. Editing operations should be performed at object level, which requires us to give each object geometric and photometric representations that are independent of the rest of the scene. Since a scene is usually acquired as a whole, this kind of object-level information is not directly available from the captured geometry or from photographs.

There are two basic problems related to this issue. First, we need to segment the scene into objects. In this paper, we use a laser range finder to acquire a discrete representation of the geometry, a point cloud. Each point in the cloud has a 3D position, an estimated normal orientation of the underlying surface at that point, and a returned laser intensity value. These cues give adequate information to distinguish points that belong to different objects. Therefore, we do segmentation on the point cloud before a mesh is actually built. Our technique is an extension of a 2D image segmentation algorithm using spectral graph theory. The result is a binary tree with individual surfaces as leaves. We first oversegment the scene into a set of coherent surfaces and then ask the user to interactively group surfaces into semantically meaningful objects.

Second, we need to attach detailed photometric properties, such as reflectance and texture, to the objects. We use a digital camera to capture such information and then recover the camera poses. In this way, we can set up correspondences between pixels in the photographs and 3D points in the scene. Using calibration targets, we developed an automatic technique for recovering camera pose for arbitrary position and orientation relative to the geometry. We efficiently search for the correct matches between the detected calibration targets in these two types of images and then solve a least-squares problem to recover the parameters of camera pose. This technique has much better average time complexity than previous algorithms [26] in the same category. With correctly registered images, space efficient texture maps can be synthesized for hardware texture-mapping.

Addressing these two problems adds much more flexibility to geometric and photometric data capture, cleans the obstacles in extracting individual objects from range and radiance images, and allows humans to interactively manipulate them. It makes it easier to build an object library based on the real world, which can be composed to form novel scenes using the objects for rendering and animation.

- *Y. Yu is with the Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Ave., Urbana, IL 61801. E-mail: yyz@cs.uiuc.edu.*
- *A. Ferencz and J. Malik are with the Computer Science Division, University of California, Berkeley, CA 94720. E-mail: {ferencz, malik}@cs.berkeley.edu.*

Manuscript received 24 July 2000; revised 7 Dec. 2000; accepted 6 Feb. 2001. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 112573.

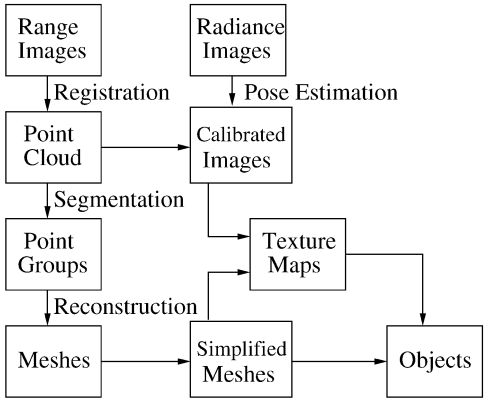


Fig. 1. **Pipeline.** This figure shows the multiple stages in our data processing procedure.

1.1 Overview

The input to our pipeline is a set of range images and photographs. The range images are registered together first to give a unified point cloud by using Cyra Technologies' software [9] to automatically locate calibration targets in each scan and interactively setting up corresponding targets among different scans. The rotation and translation between two laser scans can be recovered from three pairs of correspondences, but the more the better. The calibration targets are designed to be strongly retroreflective at the wavelength of the laser beam in order to be identified automatically.

The segmentation algorithm is then run on the point cloud, breaking it into groups. With some user interaction, we can assemble these groups into objects. We then build a mesh for each object and run mesh simplification to reduce its complexity. At the same time, the camera poses of the photographs are recovered automatically relative to the unified point cloud. To attach detailed texture information to the objects, we compose texture maps using data from multiple photographs for all the objects. In the end, we can realistically rerender the scene using the extracted geometric and photometric properties and manipulate the objects as we wish.

From this process, we can see that range data segmentation can be useful in multiple stages. Fitting smooth parametric surfaces to individual objects is much easier after segmentation and memory capacity becomes less of a limitation if we only reconstruct a mesh for one object at a time. Mesh simplification is improved since we do not intend to simplify over segmentation boundary. Segmentation into surfaces with approximately uniform specularities would aid in recovering specular reflectance models of the surfaces.

2 Previous Work

The work we present in this paper has been made possible by previous work in geometry acquisition, mesh reconstruction and simplification, image-based rendering and texture-mapping, 2D image registration and segmentation, and range image registration and segmentation.

Recent work in laser range scanning has made it possible to recover accurate geometry of real-world scenes. Besl and McKay [3] introduced the iterative closest point (ICP) algorithm to register multiple range images. Pulli [37] addressed the same problem for large data sets. A number of robust techniques for merging multiple range images into complex models are now available [7], [48], [8], [49], [35]. The authors of [22], [12], [1] also introduced techniques to a more general problem, which is mesh reconstruction from unorganized points. The reconstructed meshes from the above techniques usually have a huge amount of complexity, which makes rendering inefficient. There is a large amount of work on obtaining simplified meshes with minimal deviation in shape [23], [14], [29].

Techniques have been developed recently for texture-mapping recovered geometry [11], [10], [52], [33]. Some image-based rendering work has been introduced to implicitly make use of recovered depth information in addition to images [32], [42], [35].

There are two areas of research that are most related to this paper: segmentation and camera pose estimation.

2.1 Range Image Segmentation

There has been much work on range image segmentation in the computer vision and graphics community [4], [21], [34], [2], [24]. Most of these techniques consider every range image as a rectangular array of points with 3D positions. Due to the similarity between range images and 2D images, most of the image segmentation techniques, such as edge detection, region growing, and pixel clustering, can be applied to range image segmentation [24]. The major cues used include the depth value and normal orientation at each pixel. Most of the previous work in this area was for fitting surface primitives, such as planar and quadric patches, and generalized cylinders. Therefore, statistical tests are often incorporated into the segmentation algorithms to verify whether a particular surface primitive can fit well to a group of pixels and to determine whether region split or merge should be considered. This type of algorithm works well on range images of mechanical CAD models which consist of planar and simple curved surfaces.

However, most real objects have unknown free-form shapes, such as statues and curtains. To extract objects from a real scene, we are more interested in a segmentation algorithm that is independent of any surface primitive. Region growing approaches based on local split and merge decisions are not very appropriate, either, because the decisions made are local, which is suboptimal in finding object boundaries. We need some top-down algorithm to make global decisions. On the other hand, we may need multiple range images from different angles to capture a complex model. A segmentation algorithm, which can work on multiple registered range images simultaneously, is more interesting. In this situation, the order of points defined by the rectangular array in a single image is lost. We need new techniques to solve this more complicated problem. In the 2D image segmentation literature, some recent spectral graph theory-based algorithms [44], [36], [30] appeared to perform better than other techniques. They make decisions on where to partition the data using global information. We will extend the normalized cut framework in [44] to 3D range image segmentation in the next section.

Mangan and Whitaker [31] propose a technique for segmenting surface meshes by generalizing morphological watersheds. It is not directly relevant to the problem we are looking into because we consider segmentation as a very fundamental data processing stage which should happen at least in parallel to mesh reconstruction, not after. Effective segmentation of points into groups should be able to benefit mesh reconstruction.

2.2 Camera Pose Estimation

The mathematical foundation for pose estimation from points, lines, and curves has been extensively studied. Estimation based only on point correspondences from four [25] and more points is introduced in [13], [19]. Haralick et al. [20] provides a review of many 3-point techniques with a careful analysis of their stability. Ji et al. [39] develops an analytic least squares technique for pose estimation from points, lines, and ellipse-circle pairs. These methods all assume a known correspondence between geometry and image features, which often requires extensive user involvement.

Several techniques have been developed for automatic detection of features in the image and the geometry and finding their correspondences. Most of these techniques are applicable to discrete geometric objects whose shape is known exactly. Huttenlocher and Ullman [26] find corners and run a combinatorial search to find matches. Wunsch and Hirzinger [50] propose another method based on the iterative closest point algorithm [3]. These methods, however, are very restrictive as to the types of models they can handle, restricting themselves to simple CAD objects.

A compromise solution is to ask the user to suggest an initial pose by, for example, selecting a few point correspondences and then using object silhouettes to refine the estimation. Neugebauer and Klein [33] use this technique in addition to aligning the texture maps on the surface. They require an exact model and numerous photographs of the object, conditions we are not guaranteed.

3 Range Data Segmentation

The input data to this problem is a 3D point cloud created by merging the points from multiple registered range images. In addition to 3D position, each point has two associated attributes, a normal orientation estimated from neighboring pixels in the scan image, and a returned laser intensity value which depends mostly on the surface reflectance corresponding to the wavelength of the laser beam. The output of this module is a partition of the point cloud, treated as a set, into subsets such that each subset defines a complete object. The subsets are mutually exclusive, and their union is the complete set.

We achieve this goal in two steps. For the first step, we developed an automatic algorithm to partition the points into surface regions, each of which has approximately uniform geometric and photometric properties represented by 3D locations, surface normals, and returned laser intensities. In the second step, we interactively group surface regions into individual objects such that each object can be treated separately but points in the same object are treated in the same way. Note that surfaces in the same object may have very different surface properties. To give an example, suppose one of the objects in the scene is a cube resting on a table. The first step would return five surfaces; the user is responsible for indicating that all these surfaces belong to one object. Considerable semantic knowledge can be involved in judging whether an object is merely resting on another or is rigidly attached and is thus part of the same object; at this stage, we think it prudent to leave this judgment to a user.

In the rest of the section, we introduce the algorithm for automatic segmentation of the point set into surface regions. This is done by generalizing the normalized cut algorithm [44] to range data. There are three key issues to consider here. Namely, 1) what the appropriate similarity measure for range data is, 2) precisely what the criterion to partition the graph is, and 3) what the technique to obtain approximate solutions for large datasets is.

3.1 Normalized Cut Framework

We introduce some details of the *normalized cut* algorithm [44] here. A graph $G = (V, E)$ is defined on the input data. In our context, the nodes represent local clusters derived from the point cloud with the associated attributes. An edge in E, (s, t) with $s, t \in V$, has a weight $w(s, t)$ defined by the similarity between the location and attributes of the two nodes defining the edge. The idea is to partition the nodes into two subsets, A and B, such that the following disassociation measure, the normalized cut, is minimized.

$$Ncut(A, B) = \frac{cut(A, B)}{asso(A, V)} + \frac{cut(A, B)}{asso(B, V)}, \tag{1}$$

where $cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$ is the total connection from nodes in A to nodes in B, $asso(A, V) = \sum_{s \in A, t \in V} w(s, t)$ is the total connection from nodes in A to all nodes in the graph, and $asso(B, V)$ is similarly defined. This measure works much better than $cut(A, B)$ because it favors relatively balanced subregions instead of cutting small sets of isolated nodes in the graph.

To compute the optimal partition based on the above measure is NP-hard. However [44] shows that a good approximation can be obtained by relaxing the discrete version of the problem to a continuous one which can be solved using eigendecomposition techniques. Let y be the indicator vector of a partition. Each element of y takes two discrete values to indicate whether a particular node in the graph belongs to A or B. If y is relaxed to take on continuous real values, it can be shown that the optimal solution can be obtained by solving the generalized eigenvalue system:

$$(D - W)y = \lambda D y, \tag{2}$$

where D is a diagonal matrix with $D(i, i) = \sum_j w(i, j)$, W is the weight matrix with $W(i, j) = w(i, j)$. The eigenvector corresponding to the second smallest eigenvalue is the optimal indicator vector in real space. A suboptimal partition can be obtained by first allowing y to take on continuous real values, solving the above generalized eigenvalue system for y, and then searching a certain

number of discrete values for the best threshold to partition the real-valued elements of y into two subgroups. The two resulting subregions from this partition can be recursively considered for further subdivision. To improve efficiency, the complete graph defined by the data is usually simplified to only have edges that connect two nearby nodes. This algorithm can be used to solve different segmentation problems by choosing different edge weight $W(u,v)$ [43], [30].

3.2 Setting Up the Graph

Since we have multiple high-resolution scans, solving the graph partition problem on the original point set is impractical. For example, we have a dataset for a large room with $19\ 800 \times 800 scans$. For comparison, note that, in the application of normalized cut to image segmentation, [44] considered 200×200 images. Thus, we group nearby[1] points into clusters such that each cluster is a node. All the points within the same cluster have similar normal orientations and laser intensities. Clustering is actually carried out incrementally to minimize memory consumption. Only one of the original range scans remains in memory every time. Part of its points get integrated into existing clusters, while others create new clusters. For every point in this range scan, we check whether there are existing clusters whose centroids are within a certain small distance from the considered point, whether the difference between the average laser intensity of such an existing cluster and the intensity of the considered point is below a threshold, and whether the difference in terms of normal orientation is also below another threshold. If there is no cluster satisfying these conditions simultaneously, a new cluster is created. The number of nodes after initial clustering is reduced to about 40,000 for the room dataset. We use the averaged spatial location, normal, and returned laser intensity of each cluster as the attributes of its corresponding node in the graph. Here, the returned laser intensity is a better cue than color from photographs because it is a better approximation of surface albedo, which is lighting independent. Thus, we do not need to worry about oversegmentation due to shadows and shading effects in photographs. Local edges are set up among clusters that are within a certain distance of one another. We also set up random long-range connections among clusters to help use global context [44]. The number of random edges incident to each node is the same, $\sqrt{n}$, where n is the number of nodes in the graph. In this way, the adjacency matrix of the graph is sparse, which makes it possible to solve the problem efficiently.

The weight $w(u,v)$ over an edge (u,v) is the product of a similarity term $S(u,v)$ and a proximity term $P(u,v)$, both of which are in the form of a Gaussian distribution. $w(u,v)$ is a local measure of how likely the points (or clusters) are to belong to the same surface. $w(u,v)$ is close to 1 for points which are likely to belong together and close to 0 for points which are likely to belong to separate objects, as judged purely from local evidence.

1. To accelerate nearest point lookup, we set up a two-dimensional grid on a virtual plane with its normal set to the average normal orientation of all the input points. Each cell in the grid has a list of points that are projected into it. To look up points that are near a certain point, we only need to check the points around the cell into which that point is actually projected.

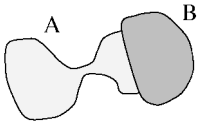


Fig. 2. Two regions, A and B, are adjacent. A correct graph partition should happen at their border, not in the middle of A.

$$S(u,v) = exp(-diff^2(u,v)/2\sigma_w^2), \tag{3}$$

where $diff(u,v)$ is the angular difference in the case of normal orientation and is the scalar difference in the case of laser intensity. Similarity in both normal orientation and laser intensity is considered during segmentation.

The proximity term over an edge (u,v) is used to model spatial coherence, which means that nearby points are more likely to belong to the same surface. Both similarity and proximity are well-known Gestalt grouping factors.

$$P(u,v) = exp(-dist^2(u,v)/2\sigma_c^2), \tag{4}$$

where $dist(u,v)$ is some distance measure. The parameters σ_w and σ_c should be set a little bit larger than the standard deviation of noise present in the input data. The standard deviation of noise can be calibrated using laser returns from a piece of flat wall with constant color. Smaller values for the parameters tend to produce oversegmented results.

Surfaces and objects at different depth should be separated in segmentation. To enhance the difference caused by depth discontinuity, we define an anisotropic distance metric $dist(u,v)$ as follows: First we define a tangential plane at each of the points u and v given the normals and positions. Let the vectors u_n and u_p be the projections of the vector $\vec{uv}$ onto the normal and tangential plane, respectively, at point u. Then,

$$dist_u(u,v) = \sqrt{E * ||u_n||^2 + ||u_p||^2/E},$$

where $E(\geq 1)$ is some adjustment parameter to magnify the difference along the normal direction and shrink the difference along the tangential direction. This parameter can make sure to segment out surfaces belonging to different layers. $dist_v(u,v)$ can be defined similarly. Finally,

$$dist(u,v) = MAX(dist_u(u,v), dist_v(u,v)). \tag{5}$$

3.3 Criterion for Graph Partition

Let us first look at the region shown in Fig. 2, where two subregions A and B are adjacent to each other. Surface attributes are uniform within the same subregion but different across subregions. Although the correct partition should happen at the boundary between A and B, normalized cut with local connections tends to partition the region in the middle of A because subregion A is quite thin in the middle and the normalized cut value in (1) is very small there. In the context of image segmentation, this can be justified as a segmentation of the image into "parts," but we felt that this was not too important a consideration for us. We therefore introduce a slight modification of the segmentation criterion by defining the *normalized weighted average cut*.

$$NWAcut(A,B) = MIN\left(\frac{WAcut(A,B)}{WAasso(A,A)}, \frac{WAcut(A,B)}{WAasso(B,B)}\right), \tag{6}$$

where

$$WAcut(A,B) = \frac{\sum_{u\in A, v\in B} S(u,v)P(u,v)}{\sum_{u\in A, v\in B} P(u,v)},$$
$$WAasso(A,A) = \frac{\sum_{u,v\in A} S(u,v)P(u,v)}{\sum_{u,v\in A} P(u,v)},$$

and $WAasso(B,B)$ is similarly defined. It is quite easy to figure out that, in Fig. 2, normalized weighted average cut is minimized at the boundary between A and B. This new measure does not favor balanced subregions as normalized cut, so it is not appropriate as the basic criterion to be used for segmentation; but, by setting a threshold on the minimum weighted average cut, we can reduce the chance of splitting in the middle of region A.

Algorithmically, we proceed as follows: For the currently considered region, first solve the sparse eigensystem (2) using Lanczos algorithm [17] to come up with a candidate partition, then check the normalized weighted average cut along the candidate boundary. If both normalized weighted average cut and normalized cut are below the threshold T_{cut}, split the region and consider the two subregions in turn; otherwise, stop recursion on the considered region.

3.4 The Complete Algorithm

We have two postprocessing steps, boundary improvement and fine segmentation, which are aimed at getting high quality segmentations, in spite of the fact that we could not process the original dataset directly in (2) because of its large size.

After a region is split into two subregions, we have an initial boundary between them. In practice, this boundary may deviate a little from the real surface boundary. To improve its localization, we exploit the linear order on the nodes in the original region according to the magnitude of their corresponding elements in the second smallest eigenvector of (2). A local segmentation problem is solved at each point cluster close to the initial boundary. At each of these clusters, collect all clusters in its neighborhood and set up a complete graph among them since the size of graph is small. Then, search for the node that best partition the linear order into two parts under normalized cut criterion. Every cluster near the initial boundary may be included into multiple local segmentations. Each local segmentation assigns the cluster to one side of the boundary or the other. We use majority voting to decide which side of the boundary it should belong to. Once we have determined the membership of all clusters, the final boundary also becomes clear.

Cues on normal orientation and laser intensity should not be applied at the same time since they have different noise levels and may interfere with and reduce each other's effectiveness during segmentation. For example, it is hard to decide where to cut first if we have two perpendicular walls with regions of distinct colors on them. In practice, segmentation is done in two passes. Continuity in normal orientation has been assigned a higher priority and is applied in the first pass. Continuity in laser intensity is applied in the second pass. But, proximity is needed in both passes to maintain spatial coherence.

To reduce the impact of the initial point clustering (which is suboptimal as it is local) at the beginning of the algorithm, we introduce an additional step at the end of the algorithm to refine segmentation results. Based on the previous segmentation, it reads in all points belonging to one group at a time, clusters them at a finer scale, and runs segmentation on the new clusters once again. The spirit of both these postprocessing steps is a coarse-to-fine refinement, something that has been tried quite extensively in various computer vision settings.

The whole segmentation algorithm is summarized as follows.

- **Coarse Segmentation:**
  - **Clustering:** Group all points into clusters such that points in the same cluster are within a prescribed radius from its centroid and have close normal orientation and laser intensity.
  - **Cluster Segmentation:**
    * Recursive segmentation based on normal continuity and proximity.
    * Recursive segmentation based on continuity in returned laser intensity and proximity.
    * **Stopping Criterion:** Both the normalized cut value and weighted average cut value are below a threshold.
    * **Boundary Improvement:** Once the stopping criterion is satisfied, apply local optimal segmentation and voting at each boundary cluster to improve boundaries.
- **Fine Segmentation:** Based on coarse segmentation results, every time only read all points that belong to one group, set smaller radius for clustering and smaller σ value in proximity term, and repeat the same steps in coarse segmentation on them.

4 Mesh Reconstruction and Simplification

Given the segmentation results on a point cloud, we can recognize the points that belong to an object and build a mesh or fit surfaces for that object. Although these are not the focus of this paper, we introduce the techniques that are applied to perform these tasks here.

There are two different methods to build a mesh for an individual object. The first method tries to extract all the points that belong to the object. This is actually a set of unorganized points since the scan order inherent in a range image is lost. We can then build a mesh using the algorithms in [22] and [1]. In practice, we use the "crust" algorithm in [1].

The crust algorithm works well for objects of which we have dense samples. However, our range images are not dense enough for objects that have fine details. So, we try to make use of the scan order in a range image and build nearest neighbor connections. In each of the original range images, we first mark the points that belong to the object and then build a connection between two points if they are

direct neighbors and both are marked. So, we can extract a partial mesh from each range image that covers the object and put them together to represent the geometry of the object. Applying the algorithms in [48] and [8] merges these partial meshes and comes up with a single mesh.

Once we have the meshes for individual objects, we continue to simplify them to improve rendering performance. Most of the previous algorithms on mesh simplification can be applied. In practice, we use the technique presented in [14]; while it is much less time-consuming to scan multiple objects simultaneously in an environment, there is little information for back-facing surfaces and surfaces that are heavily occluded. For the same reason, these surfaces are less visible and therefore less important. We interactively insert some simple polygons to model these back-facing and occluded surfaces.

5 Image Registration

Given a mesh, we are interested in texture mapping the surface from photographs. We wish to allow the user to take these photographs from any position and orientation and then effortlessly align the pictures to the geometry from range scans. More specifically, given an image and a geometry, we need to find the translation (three parameters) and the rotation (another three parameters) that describe the exact pose of the camera when the image was taken. Here, we assume that we know the internal parameters of the camera, such as focal length and radial distortion, with which we can convert the real camera into an ideal pinhole camera [13].

To calculate the pose for arbitrary rotation and translation, we need to know correspondences between features in the image and the geometry. Automatically discovering suitable features (such as lines and corners) in general scenes and matching them is extremely difficult and currently no reliable methods exist. Another possibility is to ask the user to supply the features. However, this is very labor intensive and often not very accurate (users tend to label features with an accuracy of a few pixels at best and their performance diminishes after the first dozen images). An alternative is to place unique calibration objects in the scene that are identifiable from both laser range data and images. With an ample number of such features in each image, the pose can be determined automatically and accurately without user intervention. The disadvantage of this method is that more planning must go into scene capture, ensuring that enough calibration objects are visible from each image, and that these artificial objects must then be removed from the scene. However, these limitations seem much less severe than the disadvantages of the other options.

Cyra Technologies' laser scanner which we used for this project can achieve best performance in registering multiple scans by using calibration targets taped to surfaces. We use these same targets to determine the camera pose. While these targets are specifically designed for this scanner, our techniques can be applied in general when calibration targets can be placed in the scene. These targets are flat square green patches with a white circular area in the middle. They are designed to be easily identifiable from both laser range and image data while being small to cover as little of the surface as possible. These constraints, combined with a wide variety of lighting situations that inevitably changes the apparent cover of any object, prevent us from adding a unique identifier to each target. Thus, for each image, finding the pose involves locating the targets in the image, finding the correspondences between these targets and known targets in the geometry, and, finally, calculating the six parameters of rotation and translation for the camera.

5.1 Finding Targets in Images

To find the targets in an image we first sweep target templates (white circles with green borders) of several scales over the image, using sum of squared distances (SSD) as a metric. As the circular targets actually project to elliptical patches in the image, template matching (with a liberal threshold) is only effective at locating candidate target locations (since ellipses centered at a point have three degrees of freedom, prohibitively many templates would be needed to accurately find targets using only template matching). To verify candidate regions, we attempt to fit an ellipse to the central white region and evaluate the match, again using SSD. This is equivalent to matching against the best possible elliptical template. Since the ellipse is found using a region of the image, not just a few pixels, the amount of redundancy enables us to estimate the parameters of the ellipse to subpixel accuracy. Conveniently, this technique provides five constraints for each target, which can be used to greatly reduce the combinatorial search for target correspondences.

5.2 Finding Target Correspondences

Once targets in the image have been identified, we must find their correspondence to known target locations in the geometry. This can be posed as a combinatorial search problem: Pick correspondences for enough targets to generate an overdetermined set of constraints, solve for the best pose, and test the error. If the error is within a threshold, which, in turn, is based on the expected accuracy of the point locations, accept the conjecture. If we only use the location of the center of the ellipses (x_i, y_i), without any initial guess, three correspondences is enough to find the six parameters of the pose to within four ambiguous locations, while four resolves the ambiguity and generates an overdetermined system. Unfortunately, this simple combinatorial search thus takes $O(n^4)$ time, where n is the number of targets in the geometry. Since a large scene may require a hundred or more targets (we used 66 for our room model), this search could be prohibitively expensive.

As noted above, we fit an ellipse to each target, yielding three additional parameters: major axis a_i, minor axis b_i, and rotation of the major axis to the vertical γ_i. Given a target in the geometry and an associated normal vector, we can compute the projection of the circle onto an arbitrary image plane giving a_g, b_g, γ_g. Let T_p and T_n be the position and normal of target T in the geometry and C_p the camera location. For computational convenience we only consider image planes perpendicular to the vector $Q = C_p - T_p$ (i.e., where the target is at the center of the image). For targets not located at the center of the image, we reproject them

onto such a plane and compute a_i, b_i, and γ_i relative to this new plane (this needs to be done only once for each image target). Let C_{up} be the up vector for this new image plane, r be the physical radius of the inner circle of the targets, and f the focal length of the camera. Given these, a_g, b_g, γ_g are computed by:

$$a_g = \frac{r}{\|Q\|} * f$$
$$b_g = a_g * (Q \cdot T_n)$$
$$\gamma_g = \cos^{-1}\left(\frac{T_n \times Q}{\|T_n \times Q\|} \cdot C_{up} * sign\right),$$

where

$$sign = (T_n \cdot C_{up})/|T_n \cdot C_{up}|.$$

Thus, two target correspondences provide 10 constraints (six from the two ellipses and four from the two pixel locations), which is enough to solve for a unique camera pose in the general case. We do this by anchoring the image target centers to their counterparts in the geometry, constraining the six-dimensional system to a two-dimensional manifold. We then minimize the function

$$\sum_{t=1}^{2}\Big(\big(a_{ti} - a_{tg}\big)^2 + \big(b_{ti} - b_{tg}\big)^2 + \left(\big(\gamma_{ti} - \gamma_{tg}\big) * \frac{(a_{ti} - b_{ti}) + \big(a_{tg} - b_{tg}\big)}{2}\right)^2\Big),$$

using a standard least squares optimizer. As this optimization is typically prone to a small number of local minima, we run it from multiple initial positions. A detailed treatment of pose estimation from circle/ellipse pairs can be found in [39].

Since this system with 10 equations and six unknowns is overdetermined, each solution returns an error that can be used to weed out most bad correspondences immediately. Otherwise, we use this initial guess for the pose to find a small set (typically less than three) of candidate correspondences for each remaining image target. We try these one at a time, solving the optimization for a new pose given three targets, and using the remaining targets to confirm or reject the solution.

While this still has a worst-case running time of $O(n^4)$, for any practical arrangement of targets (without many targets clumped together), we expect the running time to be $\Omega(n^2)$. As reported in the results section, this is the observed behavior, which is much faster than previous algorithms [26] whose complexity is always $O(n^4)$.

5.3 Recovering Camera Pose

Once we find an acceptable set of correspondences, we fine-tune the pose by solving a final least-squares optimization, over all six parameters, using the previous estimate as an initial position. This optimization minimizes the function:

$$\sum_{t=1}^{m}(x_{ti} - x_{tg})^2 + (y_{ti} - y_{tg})^2,$$

where (x_{tg}, y_{tg}) is the location of the projected center of target t in the image. (We do not attempt to fit a, b, γ in this function for these vary much more slowly than the projection of the center points, so they become irrelevant when enough targets are available.)

6 Texture-Mapping

For rendering and manipulation, meshes with attached texture maps are used to represent objects. Given camera poses of the photographs and the mesh of an object, we can extract texture maps for the mesh and calculate the texture coordinates of each vertex in the mesh.

We use conventional texture-mapping for the objects, which means each triangle in a mesh has some corresponding triangular texture patch in the texture map and each vertex has a pair of texture coordinates which is specified by its corresponding location in the texture map. For our situation, conventional texture-mapping is better than projective texture-mapping [11]. While texture will be projected incorrectly in projective texture-mapping once an object moves, conventional texture-mapping makes texture stick to the mesh when we move the objects. Directly mapping photographs onto a mesh using conventional texture-mapping would generate perspective distortion because photographs involve a perspective transformation and conventional texture-mapping is accurate only under affine maps. However, we can remove this kind of distortion by resampling the original photographs with a perspective transformation and warping the samples to produce new texture maps which are correct under affine maps.

Since each triangle in a mesh may be covered by multiple photographs, we actually synthesize one texture patch for each triangle to remove the redundancy. This texture patch is the weighted average of the projected areas of the triangle in all photographs. The weight for each original area from photographs is set in such a way that the weight becomes smaller when the triangle is viewed from a grazing angle or its projected area is close to the boundaries of the photograph to obtain both good resolution and smooth transition among different photographs. Visibility is determined using Z-buffer for each pixel of each original area to make sure only correct colors get averaged. We apply the scheme in [46] to place the synthetic triangular texture patches into texture maps and therefore obtain texture coordinates. This scheme quantizes the edge length (number of texels along each edge) of every texture patch to be a power of 2.

The colors for triangles invisible in all of the photographs can be obtained by propagating the colors from nearby visible triangles. This is an iterative process because invisible triangles may not have immediate neighboring triangles with colors at the very beginning. If an entire triangle is invisible, a color is obtained for each of its vertices through propagation. This color is a weighted average of the colors from the vertex's immediate neighbors with the weight in inverse proportion to their distance. If a triangle is partially visible, it is still allocated with a texture patch and the holes are filled from the boundaries inwards in the texture map. The filled colors may be propagated

(a) (b)

Fig. 3. (a) An original photograph; (b) the result of applying the derivative of the Gaussian to the image in (a). Areas with edges are allocated with more pixels during texture map synthesis.

from neighboring triangles since holes may cross triangle boundaries.

We make use of some information measure to decide the size of each texture patch so only a small number of texels are allocated for a patch without significant color variations. The response of an edge detection operator (the derivative of the Gaussian) is taken as our information measure, and applied to all original photographs (Fig. 3). For each texture patch, we use the maximum response at its corresponding pixels in the photographs to determine the number of texels it actually needs to keep the original variations on the triangle.

Additional savings of texture memory can be achieved by *reusing* texture patches for multiple 3D triangles when the texture over these triangles looks similar. For example, if the walls in a room are all white, it is possible to represent the shading variations on the walls with a small number of texture patches, even if the number of triangles for the walls is quite large. This requires that we cluster the texture patches and set the same texture coordinates to all triangles in the same cluster. The K-means algorithm (Lloyd algorithm) in vector quantization [16] can then be used to cluster all the texture patches with the same size. Because of the Mach Band effect, slight color difference along the edge shared by two 3D triangles may be rather obvious. The K-means algorithm adopts summed squared difference as its objective function. We change it slightly by allowing a distinct weight for each squared difference in the summation and set a larger weight for difference on edge texels to alleviate this effect. Given an error tolerance, we need to run a binary search to find the minimum number of clusters that can achieve that error. This process is quite time-consuming since each step of the binary search needs to run the K-means algorithm whose complexity is $O(nmd)$, where n is the number of initial vectors, m is the number of clusters, and d is the dimensionality of each vector. This complexity becomes $O(n^2d)$ when m is a large fraction of n. In practice, we exploit a two-level scheme to improve the performance by first grouping the n vectors into $\sqrt{n}$ clusters and then running the binary search on the vectors belonging to each cluster, which in turn splits into multiple clusters.

7 Results

Our algorithms have been tested on a complete real scene—a large reading room—as well as on individual laser range scans and photographs. We took 19 800 by 800 range scans of the reading room and also scanned a piano from three positions in a separate setting with Cyra Technologies' time-of-flight laser scanner [9]. Most visible surfaces were covered at centimeter accuracy. The scans were registered together using Cyra's software. A little more than 100 photographs were taken with a Sony DSC-D770 digital camera.

7.1 Geometry Segmentation and Reconstruction

Since there is noise and outliers in the scans, we filter the scans before sending them to our segmentation algorithm which runs on a Pentium II 450MHz PC. For the reading room, our segmentation code produced 393 groups in four hours which were further grouped into 95 objects and surfaces within two hours of user interaction. No user-assisted segmentation is needed. The results from segmentation are shown in Fig. 4c. A visualization of the meshes after user-assisted grouping is shown in Fig. 4d with different colors for different objects. All the curtains and furniture, including lamps, tables, couches, dressers, and chairs, are correctly segmented out. Before user interaction, the number of groups for each object ranges from two to 20, with an average of four. Oversegmentation produced extra groups on walls and along object boundaries, most of which are not very visually noticeable. Some of them are marked out with white circles in Fig. 4c. We did not group oversegmented pieces on walls together interactively because we do not have the need to manipulate a complete piece of wall. Fig. 4a and Fig. 4b also give the segmentation results for a different, but simpler, room and a single facade. In Fig. 4a, a person was sitting in the next room

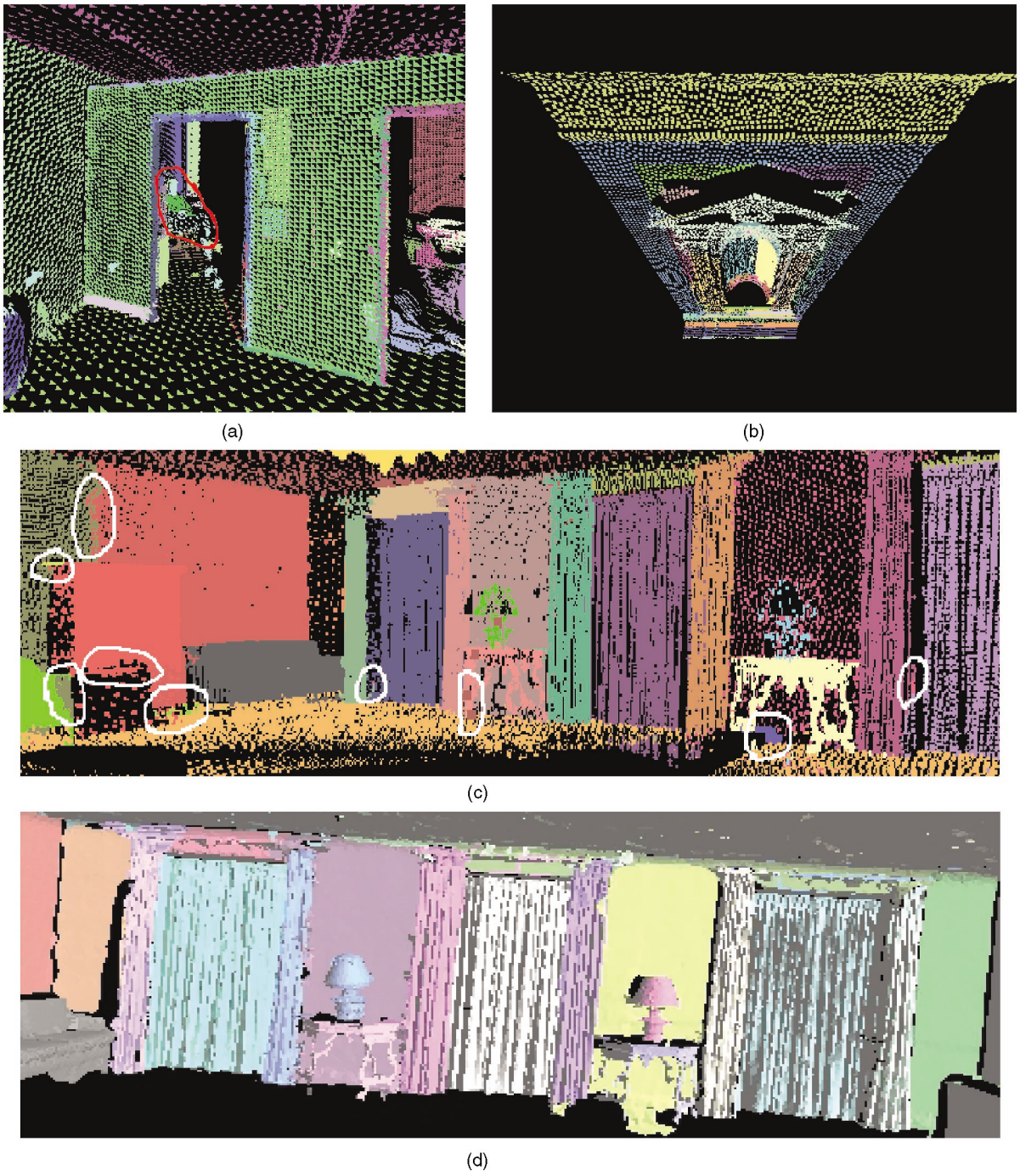


Fig. 4. The segmentation results on three data sets: (a) A simple room with portals, (b) Albert Hall facade, (c) a large reading room. (a), (b), and (c) show the segmentation results from our algorithm. Each dot represents one point cluster. Clusters in the same group are shown with the same color and density. Some oversegmentations are marked out with white circles in (c). (d) Shows the segmentation results as multiple pieces of colored meshes for the reading room with furniture and curtains after the user interactively grouped surfaces into objects. Objects are shown in distinct colors.

while it was scanned. We can see his head and torso (in the region marked out with a red line) are correctly segmented out from the rest of the environment. Fig. 4b shows that our anisotropic distance metric in (5) can effectively separate layers at different depth.

Most of the meshes, including the piano, were reconstructed using the "crust" algorithm [1]. Antique tables and chairs, as well as curtains, were reconstructed using an algorithm similar to [8]. Fig. 5 shows an image of the meshes which includes lamps, tables, curtains, couches, as well as the ceiling and walls. To demonstrate that we really have extracted individual objects, Fig. 6 shows the individual models of an antique table, a chair, and the piano. The points on the floor were segmented out automatically and a single plane was fit to replace those points.

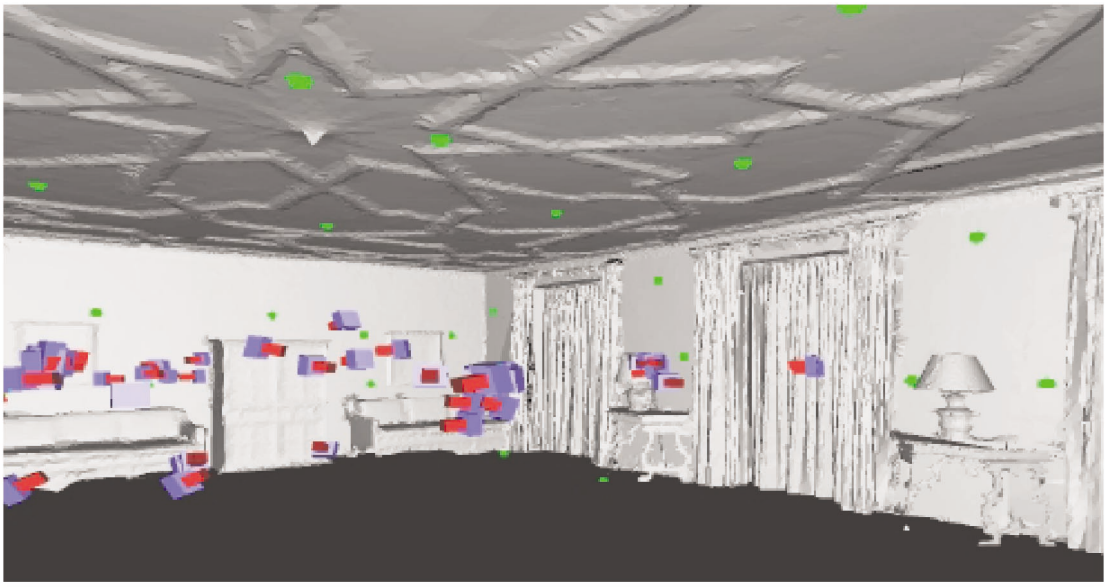
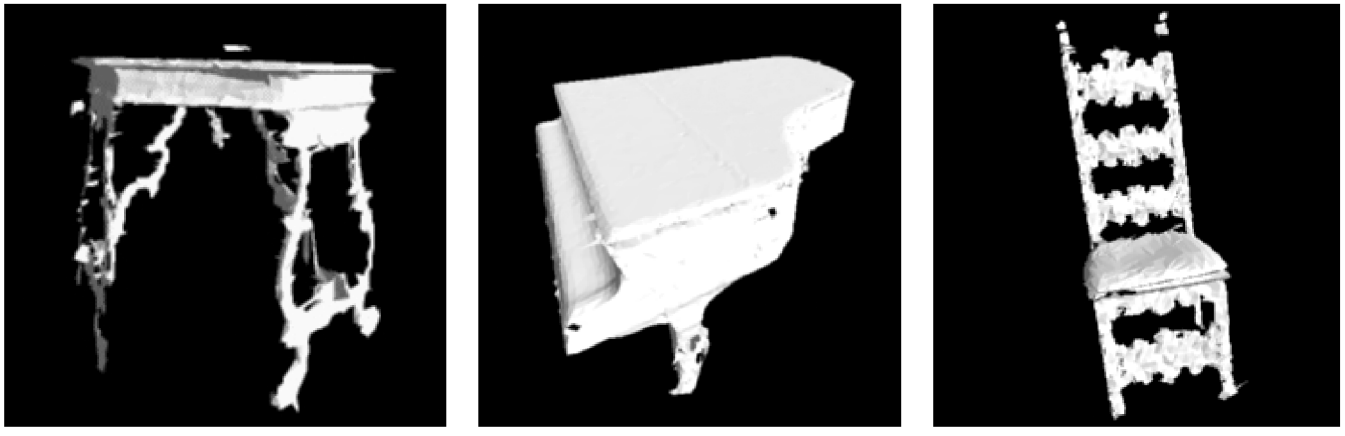


Fig. 5. The reconstructed meshes with targets (green) and recovered camera poses (red and blue) for the reading room. There is a separate mesh for each object.



(a) (b) (c)

Fig. 6. Geometric models of three objects. The meshes were built after their corresponding points were segmented out from the range images using our segmentation algorithm. (a) An antique table. (b) A Piano. (c) An antique chair.

7.2 Camera Pose Estimation

To evaluate our camera pose estimation technique, we look at three aspects: 1) the amount of user intervention, 2) the accuracy of the resulting pose, and 3) the computational cost of our algorithm. We ran our automatic algorithm on 62 images with four or more targets visible. In these images, the automatic target detector found 90 percent of the visible targets, while finding four false matches. These errors were easily correctable by prompting the user to localize the search. Poses were estimated correctly for 58 of the 62 images. The remaining four coincidentally lined up with an erroneous set of targets in the geometry. These errors could also be easily corrected interactively by supplying one pair of correspondences. The amount of user intervention was approximately 15 minutes.

We found the accuracy of our estimated pose to be very high, typically within one pixel. An example of this is shown in Fig. 7, where a sample image is texture-mapped onto the geometry and the resulting surfaces are displayed from a different viewing direction (black areas in Fig. 7c indicate backfacing or occluded areas in the geometry). Note the object boundaries in the image line up with geometric discontinuities in the scene.

As expected, our algorithm runs in $O(n^2)$ time for real-world inputs. For 50 targets in the geometry, the running time was 5.8 seconds, while, for 100 targets, the algorithm took 21 seconds.

From those calibrated camera poses and simplified meshes, we synthetically composed 22 1,024 by 512 texture maps that are used to render the original as well as the altered scene.

7.3 Scene Editing

Our ultimate goal is object manipulation. Fig. 8 shows two comparisons to demonstrate our ability to do scene editing. The images on the left are rerenderings of the original scene from a novel viewpoint by texture-mapping the reconstructed geometric models. The images on the right show

Fig. 7. (a) A real photograph with targets located. (b) The scanned point cloud viewed from the recovered camera pose. (c) Low-resolution texture-mapping using a single photograph. Note the edge alignment between the image and the geometry.

synthetically composed scenes. In Fig. 8b, a couch is moved and replicated to places near the fireplace, a piano inserted and placed near where the couch was. In Fig. 8d, we can see two lamps flying in the air. Fig. 9 gives two more images with novel scene compositions.

8 Conclusions and Future Work

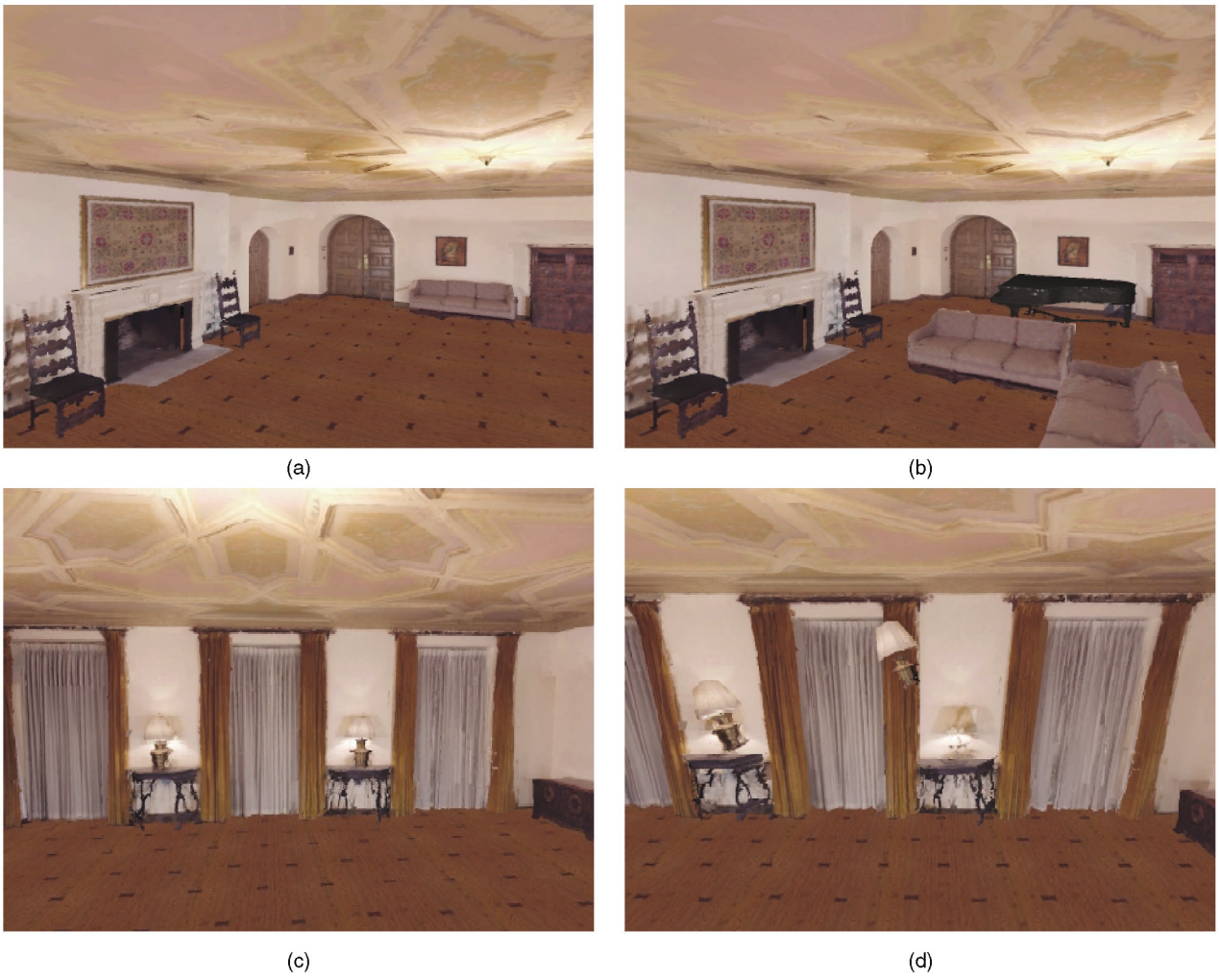
We presented two automatic techniques, range data segmentation and camera pose estimation, that are necessary for building an object-level representation and scene editing for a real scene captured with both cameras and laser range scanners. Range data segmentation enables building separate geometric models for each individual object in the scene. Camera pose estimation enables accurate alignment between photographs and geometry, which in turn makes texture-mapping individual objects possible.

A multistage data processing pipeline is proposed for building such an object-level representation. In this pipeline, the stage of building a high-quality mesh for each object remains challenging. This is because only incomplete range data can be obtained for objects present in a large environment due to accessibility and occlusion. Fitting smooth models to local regions seems a promising approach to filling in the missing parts.

Some other stages in the pipeline also need improvement. One should be able to recover surface reflectance from photographs using the techniques in [40], [51] to obtain a lighting independent representation of each object. Surface reflectance also gives more information for geometry segmentation and image registration. However, recovering surface reflectance information itself requires segmentation and image registration. So, it may be possible to improve these three types of estimation simultaneously in an iterative framework.

Acknowledgments

The authors would like to thank Ben Kacyra, Mark Wheeler, Daniel Chudak, and Jonathan Kung at Cyra Technologies, Inc. for their help and advice in using their time-of-flight laser scanner and the accompanying Cyrax software. Thanks to Marc Levoy, Brian Curless, and Lucas Pereira



(a) (b)

(c) (d)

Fig. 8. (a) and c) Synthetic images with objects in their original positions rendered using texture-mapping. (b) and (d) Synthetic images with object insertion and relocation. A piano is added to the room, a couch and two lamps moved and replicated.

(a) (b)

Fig. 9. (a) and (b) More synthetic images with object manipulation.

for providing us with their volumetric mesh reconstruction software during an early testing stage of this project, Jianbo Shi for some helpful discussions on normalized cuts, Johnny Chang for his help on demonstrations during the preparation of an earlier version of this paper, and the reviewers for their valuable comments. Mesh simplification was done with the software from Michael Garland. This research was sponsored by a Multidisciplinary University

Research Initiative on 3D direct visualization from the US Office of Naval Research and BMDO, grant FDN00014-96-1-1200, the California MICRO program, and the Microsoft Graduate Fellowship program.

REFERENCES

[1] N. Amenta, M. Bern, and M. Kamvysselis, "A New Voronoi-Based Surface Reconstruction Algorithm," *Proc. SIGGRAPH '98,* pp. 415-421, 1998.

[2] A. Leonardis, A. Gupta, and R. Bajcsy, "Segmentation of Range Images as the Search for Geometric Parametric Models," *Int'l J. Computer Vision,* vol. 14, no. 3, pp. 253-277, 1995.

[3] P.J. Besl and N.D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 5, pp. 239-256, May 1992.

[4] P.J. Besl and R.C. Jain, "Segmentation through Variable-Order Surface Fitting," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 10, no. 2, pp. 167-192, Feb. 1988.

[5] J.-Y. Bouguet and P. Perona, "3D Photography on Your Desk," *Proc. Int'l Conf. Computer Vision (ICCV),* 1998.

[6] E. Chen, "QuickTime VR—An Image-Based Approach to Virtual Environment Navigation," *Proc. SIGGRAPH '95,* 1995.

[7] Y. Chen and G. Medioni, "Object Modeling from Multiple Range Images," *Image and Vision Computing,* vol. 10, no. 3, pp. 145-155, Apr. 1992.

[8] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Proc. SIGGRAPH '96,* pp. 303-312, 1996.

[9] Cyra Technologies, Inc., Online Documents, www.cyra.com/cyrax.html, year?

[10] P. Debevec, Y. Yu, and G. Borshukov, "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping," *Proc. Ninth Eurographics Workshop Rendering,* pp. 105-116, 1998.

[11] P.E. Debevec, C.J. Taylor, and J. Malik, "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach," *Proc. SIGGRAPH '96,* pp. 11-20, Aug. 1996.

[12] H. Edelsbrunner and D.P. Mücke, "Three-Dimensional Alpha Shapes," *ACM Trans, Graphics,* vol. 13, pp. 43-72, 1994.

[13] O. Faugeras, *Three-Dimensional Computer Vision.* Cambridge, Mass.: MIT Press, 1993.

[14] M. Garland and P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics," *Proc. SIGGRAPH '97,* pp. 209-216, 1997.

[15] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 6, no. 6, pp. 30-50, 1984.

[16] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression.* Boston: Kluwer Academic, 1992.

[17] ? Golub and ? Van Loan, *Matrix Computations.* John Hopkins Press, 1989.

[18] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen, "The Lumigraph," *Proc. SIGGRAPH '96,* pp. 43-54, 1996.

[19] R.M. Haralick, H. Joo, C.-N. Lee, X. Zhuang, and M.B. Kim, "Pose Estimation from Corresponding Point Data," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 19, no. 6, p. 1426, 1989.

[20] R.M. Haralick, C. Lee, K. Ottenberg, and M. Nolle, "Review and Analysis of Solutions of the Three Point Perspective Pose Estimation," *Int'l J. Computer Vision,* vol. 13, no. 3, pp. 331-356, 1994.

[21] R.L. Hoffman and A.K. Jain, "Segmentation and Classification of Range Images," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 9, no. 5, pp. 608-620, May 1987.

[22] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Proc. SIGGRAPH '92,* pp. 71-78, 1992.

[23] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh Optimization," *Proc. SIGGRAPH '93,* pp. 19-26, 1993.

[24] A. Hoover, G. Jean-Baptiste, X.Y. Jiang, P.J. Flynn, H. Bunke, D.B. Goldgof, K. Bowyer, D.W. Eggert, A. Fitzgibbon, and R.B. Fisher, "An Experimental Comparison of Range Image Segmentation Algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence,* pp. 673-689, 1996.

[25] Y. Hung, P.S. Yeh, and D. Harwood, "Passive Ranging to Known Planar Point Sets," *Proc. IEEE Int'l Conf. Robotics and Automation,* pp. 80-85, 1985.

[26] D.P. Huttenlocher and S. Ullman, "Recognizing Solid Objects by Alignment with an Image," *Int'l J. Computer Vision,* vol. 5, no. 2, pp. 195-212, 1990.

[27] S. Laveau and O. Faugeras, "3-D Scene Representation as a Collection of Images," *Proc. 12th Int'l Conf. Pattern Recognition,* vol. 1, pp. 689-691, 1994.

[28] M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. SIGGRAPH '96,* pp. 31-42, 1996.

[29] P. Lindstrom and G. Turk, "Evaluation of Memoryless Simplification," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 2, pp. 98-115, Apr.-June 1999.

[30] J. Malik, S. Belongie, J. Shi, and T. Leung, "Textons, Contours and Regions: Cue Combination in Image Segmentation," *Proc. Int'l Conf. Computer Vision,* 1999.

[31] A.P. Mangan and R.T. Whitaker, "Partitioning 3D Surface Meshes Using Watershed Segmentation," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 4, pp. 308-321, Oct.-Dec. 1999.

[32] L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. SIGGRAPH '95,* 1995.

[33] P.J. Neugebauer and K. Klein, "Texturing 3D Models of Real World Objects from Multiple Unregistered Photographic Views," *Proc. Eurographics '99,* pp 245-256, 1999.

[34] T.S. Newman, P.J. Flynn, and A.K. Jain, "Model-Based Classification of Quadric Surfaces," *CVGIP: Image Understanding,* vol. 58, no. 2, pp 235-249, 1993.

[35] L. Nyland et al., "The Impact of Dense Range Data on Computer Graphics," *Computer Vision and Pattern Recognition MVIEW '99,* 1999.

[36] P. Perona and W.T. Freeman, "A Factorization Approach to Grouping," *Proc. European Conf. Computer Vision '98 (ECCV '98),* pp. 655-670, 1998.

[37] K. Pulli, "Multiview Registration for Large Data Sets," *Proc. Int'l Conf. 3D Digital Imaging and Modeling,* pp. 160-168, 1999.

[38] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C.* New York: Cambridge Univ. Press, 1988.

[39] Q. Ji, M. Costa, R. Haralick, and L. Shapiro, "An Integrated Linear Technique for Pose Estimation from Different Features," *Int'l J. Pattern Recognition and Artificial Intelligence,* June 1999.

[40] Y. Sato, M.D. Wheeler, and K. Ikeuchi, "Object Shape and Reflectance Modeling from Observation," *Proc. SIGGRAPH '97,* pp. 379-387, 1997.

[41] S.M. Seitz and C.R. Dyer, "Photorealistic Scene Reconstruction by Voxel Coloring," *Computer Vision and Pattern Recognition,* pp. 1067-1073, 1997.

[42] J. Shade, S. Gortler, L.-W. He, and R. Szeliski, "Layered Depth Images," *Proc. SIGGRAPH '98,* pp. 231-242, 1998.

[43] J. Shi and J. Malik, "Motion Segmentation and Tracking Using Normalized Cuts," *Proc. Int'l Conf. Computer Vision,* 1998.

[44] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 1997.

[45] H.-Y. Shum and L.-W. He, "Rendering with Concentric Mosaics," *Proc. SIGGRAPH '99,* pp. 299-306, 1999.

[46] M. Soucy, G. Godin, and M. Rioux, "A Texture-Mapping Approach for the Compression of Colored 3D Triangulations," *Visual Computer,* vol. 12, pp. 503-514, 1996.

[47] R. Szeliski, "Image Mosaicing for Tele-Reality Applications," *IEEE Computer Graphics and Applications,* 1996.

[48] G. Turk and M. Levoy, "Zippered Polygon Meshes from Range Images," *Proc. SIGGRAPH '94,* pp. 311-318, 1994.

[49] M.D. Wheeler, Y. Sato, and K. Ikeuchi, "Consensus Surfaces for Modeling 3D Objects from Multiple Range Images," *Proc. DARPA Image Understanding Workshop,* 1997.

[50] P. Wunsch and G. Hirzinger, "Registration of CAD-Models to Images by Iterative Inverse Perspective Matching," *Proc. Int'al Conf. Pattern Recognition (ICPR),* pp. 77-83, 1996.

[51] Y. Yu, P. Debevec, J. Malik, and T. Hawkins, "Inverse Global Illumination: Recovering Reflectance Models of Real Scenes from Photographs," *Proc. SIGGRAPH '99,* pp. 215-224, July 1998.

[52] Y. Yu, "Efficient Visibility Processing for Projective Texture-Mapping," *J. Computers & Graphics,* vol. 23, no. 2, pp. 245-253, 1999.

[53] Y. Yu and J. Malik, "Recovering Photometric Properties of Architectural Scenes from Photographs," *Proc. SIGGRAPH '98,* pp. 207-217, July 1998.

Yizhou Yu received the PhD degree in computer science from the University of California at Berkeley in 2000 and the MS degree in applied mathematics and the BS degree in computer science from Zhejiang University, China, in 1994 and 1992, respectively. He is currently an assistant professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. During 1994-1996, he was a PhD student at the University of California at Santa Barbara. He has done research in computer graphics and vision, including image-based modeling and rendering, image processing, texture synthesis and mapping, visibility processing, radiosity, and global illumination, and has authored or coauthored more than 20 research papers. He is a recipient of 1998 Microsoft Graduate Fellowship and 1992 Chinese Computerworld Scholarship. His current interests include appearance modeling, computer animation, and computer vision problems with HCI applications.

Andras Ferencz studied computer science at Cornell University, graduating with the BS degree in 1997. He spent the next year in the Image Understanding Area of Xerox's Palo Alto Research Center. Since 1998, he has been a PhD student in the Computer Science Division of the University of California at Berkeley. He is a student member of the IEEE.

Jitendra Malik received the BTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1980 and the PhD degree in computer science from Stanford University in 1985. In January 1986, he joined the faculty of the Computer Science Division, Department of EECS, University of California at Berkeley, where he is currently a professor. His research interests are in computer vision and computational modeling of human vision. His work spans a range of topics in vision, including image segmentation and grouping, texture, stereopsis, and object recognition. His research has found applications in image based modeling and rendering, content-based image querying, and intelligent vehicle highway systems. He has authored or coauthored more than 90 research papers on these topics. He received the gold medal for the best graduating student in Electrical Engineering from IIT Kanpur in 1980, a Presidential Young Investigator Award in 1989, and the Rosenbaum fellowship for the Computer Vision Programme at the Newton Institute of Mathematical Sciences, University of Cambridge in 1993. He received the Diane S. McEntyre Award for Excellence in Teaching from the Computer Science Division, University of California at Berkeley, in 2000. He is an editor-in-chief of the *International Journal of Computer Vision*.