Acquisition and Visualization of Colored 3D Objects

Kari Pulli
Stanford University
Stanford, CA, U.S.A
kapu@cs.stanford.edu

Habib Abi-Rached, Tom Duchamp, Linda G. Shapiro and Werner Stuetzle
University of Washington
Seattle, WA, U.S.A
{ habib@ee | duchamp@math | shapiro@cs | wxs@stat }.washington.edu

Abstract

This paper presents a complete system for scanning the geometry and surface color of a 3D object and for displaying realistic images of the object from arbitrary viewpoints. A stereo system with active light produces several views of dense range and color data. The data is registered and a surface that approximates the data is constructed. The surface estimate can be fairly coarse, as the appearance of fine detail is recreated by view-dependent texturing of the surface using color images.

1. Introduction

Convincing virtual reality environments require realistic object models, but such models are often difficult to construct synthetically. We have built a semi-automatic system that can acquire range data, register it, construct a model that incorporates both geometry and color, and render it from an arbitrary viewpoint. In this paper, we describe the system, emphasizing view-dependent texturing of geometric models.

2. 3D object reconstruction

3D object reconstruction consists of three steps: data acquisition, registration, and surface reconstruction.

2.1. Data acquisition

In order to keep our system inexpensive and to better control the acquisition process, we built our own scanner for acquiring range and color data. Our scanner consists of four digital color cameras and a slide projector sitting on a computer-controlled turntable (see Fig. 1). The slide projector emits a vertical stripe of white light into the working volume.

The range data is obtained through triangulation. In an otherwise dark room, the vertical light stripe illuminates the scene and the objects, and appears in the camera images as shown in Fig. 2. The illuminated pixels in the center of the stripe are paired with pixels in the other images by intersecting the epipolar line (shown dotted) of a pixel in the left image with the image of the stripe in the right image. Triangulation yields the 3D coordinates of the surface point corresponding to the paired pixels. The whole scene can be digitized a stripe at a time, by sweeping the stripe across the scene in small steps. Finally, we take a color image. Background pixels are determined by back lighting objects and tracking which pixels change color and intensity.

Figure 1. The scanner hardware.

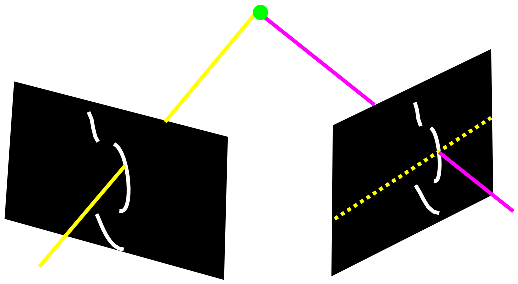

Figure 2. Range through triangulation.

Accurate triangulation requires accurately locating the center of the stripe in a camera image. However, the intensity distribution across the width of the stripe is approximately Gaussian only if the illuminated surface is locally planar and the whole width of the stripe is visible to the cam-

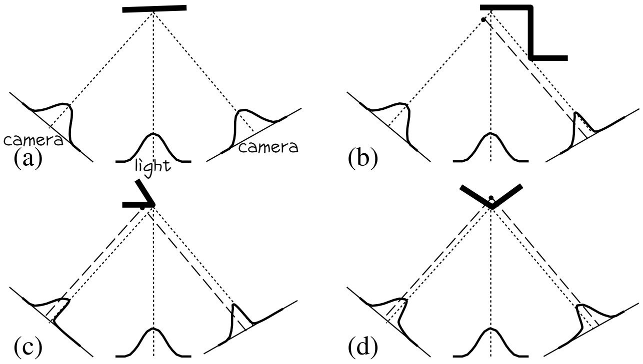

Figure 3. Error sources in range by triangulation.

era. Curless and Levoy [2] noticed that with more complicated geometry the range estimates become systematically distorted. Figure 3 (adapted from [2]), shows three sources of error in estimating range by triangulation.

Figure 3(a) shows the ideal situation. In Fig. 3(b) the beam is partially occluded in the right camera, and the estimate of the stripe center is shifted to the left. In Fig. 3(c) the beam is aimed at a sharp silhouette corner of the object, and only half of the beam hits the surface. Both stripe center estimates are shifted to the left. Figure 3(d) shows the beam at a crease on the surface. Both the right and the left beams are foreshortened.

To solve these problems, we implemented an algorithm for locating centers of stripes based on *spacetime analysis* [2]. Our algorithm reduced both the magnitude and the frequency of the errors illustrated in Fig. 3. Let us assume that the beam is wide enough to cover several pixels in the image, that its intensity has Gaussian distribution, and that we move the beam in steps that are small compared to the beam width. As the beam sweeps past the point on the surface that corresponds to a given pixel, the intensity of the pixel first increases and then decreases. Because the shape of the time-varying intensity profile is always Gaussian, the time when the beam was centered on the pixel can be reliably estimated from that profile.

Spacetime analysis assigns to each pixel in each image the time at which the beam was centered on it. We can use this information to find a corresponding pixel in the right image for each pixel in the left image as follows. Choose a pixel in the left image and note the time when the beam was centered at that pixel. The epipolar line of that pixel in the right image is parameterized by the time when the pixels under the line were illuminated by the beam. The image location corresponding to the original pixel is found in subpixel accuracy by finding the location on the line that corresponds to the time associated with the original pixel. The 3D coordinates of the surface point are found by triangulation as before.

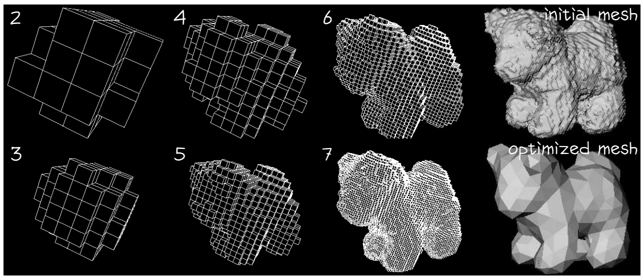

Figure 4. The results of hierarchical space carving after 2 to 7 octree subdivisions. On the right: the smoothed initial surface, and the result after mesh optimization.

2.2. Registration

To obtain complete coverage of the an object we have to perform scans from several viewpoints. Between scans, the object is moved. Because the range data in each view is expressed in the sensor coordinate system, to estimate a surface from the data, and to project the color images to that surface, we have to align, or *register*, the views. To register a view to another one, we obtain an approximate solution by interactively matching features and aligning the views in 3D. Then the colored range data of one view is projected onto the color image of the other one, the color images are aligned, and the 3D data points that project to the same pixel are paired. Using the paired points a rigid transformation is found that aligns most of the pairs. The process is iterated until convergence in the same manner as in the Iterated Closest Points method [1]. Multiview registration is handled by first registering views pairwise, determining and storing a set of reliable point pairs, and finally simultaneously minimizing the distances between all the stored point pairs. More detail for both pairwise and multiview registration can be found in [6].

2.3. Surface reconstruction

Once the data is registered to a common coordinate system, we construct a model of the surface of the scanned object through a two-phase process. We create an initial surface estimate using a hierarchical space carving method [8]. Space carving relies on the fact that if the scanner can observe a surface, the space between the surface and the scanner is empty. The space is tessellated into cubes, and if there is a view (a range scan) such that the cube is either in front of the data or projects onto the background, the cube is removed. The space carving is done hierarchically using an octree. 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 remove. If a cube cannot be removed (and is not behind the surface in every view) it is recursively subdivided and tested.

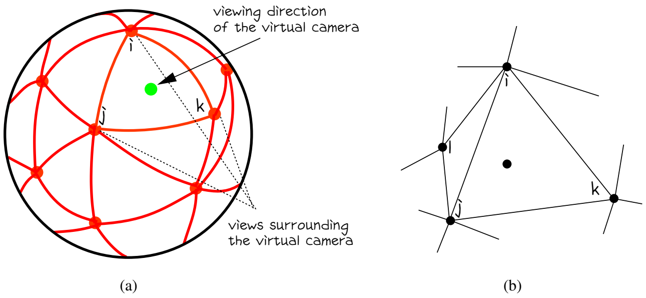

Figure 5. (a) The triangle containing the viewing direction of the virtual camera determines the three views used to search for candidate rays. (b) Though view l is closer to the current viewing direction, view k is a better choice.

The surface estimate consists of the free faces of the remaining cubes. The initial mesh is simplified and better fitted to the data using the mesh optimization algorithm by Hoppe *et al.* [5]. The surface reconstruction process is illustrated in Fig. 4.

3. View-dependent texturing

The model of the surface that we construct is displayed by a view-dependent texturing algorithm that uses the original color images. Rather than calculating a texture map that is pasted onto the surface, we use the surface geometry to synthesize new color images from the original input images. In the following we first describe how the input images used to synthesize a new image are chosen. Then we explain how the pixels in the input images that correspond to the object surface location visible to a particular pixel in the viewer are found. We describe our averaging scheme for combining those rays, and we finally discuss the preprocessing steps that allow an interactive implementation of our texturing method.

3.1. Choosing views

In principle, any camera view that sees the same surface point as the viewer (a virtual camera) could contribute to the color of the corresponding 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 and only a small portion of the surface, if any, is likely to be visible both to the viewer and to the distant view. Additionally, small errors in registration, camera calibration, and surface reconstruction lead to larger errors in backprojecting surface points to the color images. In our implementation we only search for compatible rays from three input images that have been taken from nearby viewing directions.

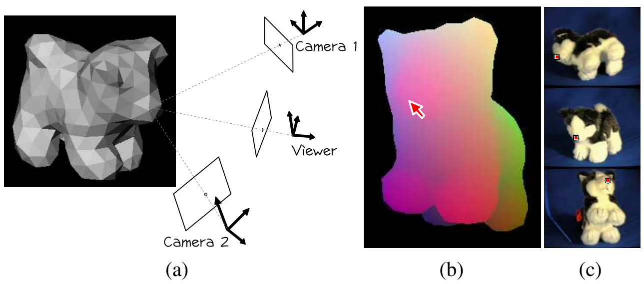

Figure 6. (a) A ray from the viewer is cast through a pixel, intersecting the object surface, and is projected back to color images, producing candidate pixels for coloring the original pixel. (b) A false color rendering of the surface geometry is used to find the surface point visible through each pixel. (c) The 3D point corresponding to the pixel pointed to in (b) is projected into three color images.

To facilitate the selection of suitable views, the views are organized as illustrated in Fig. 5(a). Our algorithm places a vertex corresponding to each viewing direction of the input images on a unit sphere, and then computes a Delaunay triangulation of the sphere using those vertices. When rendering a frame, an extra vertex corresponding to the current viewing direction is placed on the unit sphere as shown in Fig. 5(a). The triangle containing that vertex determines the three views within which the algorithm will search for candidate rays for the surface points visible to the viewer. Note that these views are not always the three closest views, though the closest one is guaranteed to be among the three. For example, in Fig. 5(b) view l is closer to the current view direction than view k. However, we prefer to use view k because i, j, and l all lie to the “left” 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.

3.2. Finding compatible rays

When our viewer is aimed at an object, the first task in determining the color of a particular pixel is to locate the point on the object surface that is visible through that pixel. Figure 6(a) shows a ray through one of the viewer’s pixels ending at its first intersection with the object surface. Candidate rays that might see the same surface point are obtained by projecting the intersection point back to the input images. For example, the viewer pixel marked by the arrow in Fig. 6(b) corresponds to a point on the dog’s snout, which projects back to the dots displayed in the three images in Fig. 6(c).

We can use graphics hardware to determine the surface point visible through a given pixel. The method (also used by Gortler *et al.* [4]) is illustrated in Fig. 6(b). First, the axis-aligned bounding box for the triangle mesh representing the

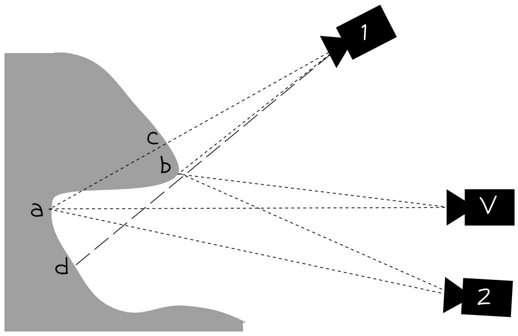

Figure 7. The virtual camera sees two points a and b, but they project back to pixels of camera 1 that actually see points c and d.

object is calculated. Then the coordinates of each vertex are scaled and translated so that the bounding box becomes a cube with unit-length sides. Now the x, y, and z coordinates of each vertex can be encoded in the red, green, and blue components of its color, so that when the mesh is rendered in the viewer, an image like the one in Fig. 6(b) is produced. 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. A slightly slower alternative would be to consult the z-buffer to determine the surface coordinates.

Once the surface point corresponding to a viewer pixel has been determined, candidate rays are obtained by projecting that point back to the input images as shown in Fig. 6(c). To perform these projections, each camera's internal and external parameters must be known. In our case, the internal parameters are obtained from camera calibration parameters; the external parameters are obtained by registering the range maps into a common coordinate system.

As Fig. 7 illustrates, not all the candidate rays obtained through backprojection should be accepted. The virtual camera of the viewer sees two surface points, a and b, and those points are also clearly visible to camera 2. However, point a is not visible to camera 1 due to self-occlusion; the ray from camera 1 pointing at a sees point c instead. Point b, on the other hand, *is* visible to camera 1, though just barely, but in this case minute errors in calibration, registration, or surface reconstruction lead point b to project to a pixel that really sees d instead. We can detect these problems easily if we retain the original range maps for each camera. For example, we can calculate the distance from point a to camera 1 and compare it to the range map value for the pixel a projects to. If these distances differ significantly (as they do in this case), then the ray is rejected.

3.3. 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.

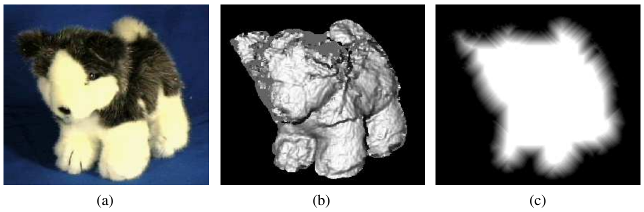

Figure 8. (a) A view of a toy dog. (b) The sampling quality weight. (c) The feathering weight.

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. Our algorithm uses the *barycentric coordinates* of the current viewing direction with respect to the directions of the three surrounding views as the directional weight. The barycentric coordinates linearly interpolate the three points to produce the fourth. In our case the points lie on a sphere rather than a plane, but the barycentric coordinates can still be computed by radially projecting the vertex of the current view direction onto the planar triangle formed by the surrounding three views.

The sampling quality weight directly reflects how well a ray samples the surface. Our algorithm assigns 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. 8(b) for a view of the toy dog. The feathering weight is used mostly to hide artifacts due to differences in lighting among the input images. Without the feathering weight, the silhouettes of the input views cause noticeable discontinuities in coloring as a view contributes to pixel colors on one side of a silhouette edge but not on the other. As illustrated in Fig. 8(c), the feathering weight is zero outside of the object, and it grows linearly to a maximum value of one within the object.

3.4. Precomputation for run-time efficiency

The directional weight changes every time the viewer moves with respect to the object, so it must be recomputed for each frame. However, since the sampling quality and feathering weights remain constant, we preprocess each pixel of each image by storing the product of those two weights 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 and a large part of that time is spent in correcting for the cylindrical lens

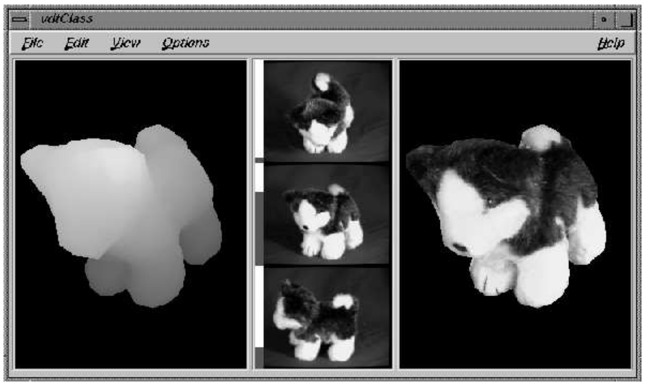

Figure 9. Our interactive viewer. Left: the colors code the visible surface points. Middle: the three views that have been chosen as inputs, along with bars that show the directional weight. Right: the final image.

distortion. We avoid this calculation by preprocessing the input images (along with associated information such as the weights and range data) to remove the distortions beforehand. The projection is further optimized by collapsing each view's registration and projection transformations into a single 3×4 matrix that transforms a homogeneous 3D surface point into a homogeneous 2D image point.

3.5. Results and discussion

We have implemented an interactive viewer for displaying view-dependently textured objects (see Fig. 9). 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 an 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 (NURBS, subdivision surfaces, etc.) by finding the visible surface points using the z-buffer instead of rendering triangles colored by location.

The weighting scheme is the same as in view-based rendering (VBR) [7]. In VBR each range scan is modeled separately as a textured mesh and several meshes are composited during rendering. However, if the object has depth discontinuities, we cannot get any range data for many pixels (we can only triangulate surface points visible to both cameras and the light source), and although the color information is valid, it cannot be used. In the current method the range data is first combined into a single surface model over which the color data is projected. This means that all valid color data can contribute to coloring a surface location even where no range value was recovered, as long as the surface was modeled using either data from another viewpoint or by space carving. In practise, the new method produces better renderings than VBR with the same input data.

Our view-dependent texturing is also related to the work of Debevec *et al.* [3], but exact comparison is not possible as no implementation details were given in that work.

4. Conclusions

We have described a complete system for scanning and displaying realistic images of colored objects. The output of our stereo system with active lighting was considerably improved by adapting spacetime analysis from [2] to our system. The data are registered into a common coordinate system, and an initial surface estimate created by hierarchical space carving is simplified and better fitted using a mesh optimization algorithm. Realistic images from arbitrary viewpoints are interactively displayed using our view-dependent texture mapping method.

5. Acknowledgements

This research was supported by grants from NSF (IRI-9520434 and DMS-9402734), Academy of Finland, and Human Interface Technology Laboratory.

References

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Patt. Anal. Machine Intell.*, 14(2):239–256, Feb. 1992.

[2] B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 987–994, June 1995.

[3] 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*, pages 11–20. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[4] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, Aug. 1993.

[6] K. Pulli. *Surface reconstruction and display from range and color data*. PhD thesis, Dept. of Computer Science and Engineering, Univ. of Washington, Dec. 1997.

[7] 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*, June 1997.

[8] 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*, pages 205–211, May 1997.