Interactive Stereoscopic Rendering of Volumetric Environments

Ming Wan, Nan Zhang, Huamin Qu, and Arie E. Kaufman, *Fellow*, *IEEE*

Abstract—We present an efficient stereoscopic rendering algorithm supporting interactive navigation through large-scale 3D voxel-based environments. In this algorithm, most of the pixel values of the right image are derived from the left image by a fast 3D warping based on a specific stereoscopic projection geometry. An accelerated volumetric ray casting then fills the remaining gaps in the warped right image. Our algorithm has been parallelized on a multiprocessor by employing effective task partitioning schemes and achieved a high cache coherency and load balancing. We also extend our stereoscopic rendering to include view-dependent shading and transparency effects. We have applied our algorithm in two virtual navigation systems, flythrough over terrain and virtual colonoscopy, and reached interactive stereoscopic rendering rates of more than 10 frames per second on a 16-processor SGI Challenge.

Index Terms—3D voxel-based environment, stereoscopic rendering, ray casting, 3D warping, splatting, antialiasing, virtual flythrough, virtual colonoscopy.

1 Introduction

Stereoscopic displays play an important role in systems that perform flexible navigation through 3D voxel-based virtual environments (in short, volumetric environments), such as an interactive virtual navigation inside CT-scanned human organs [10]. In such a system, we often generate two images of a scene that differ in their horizontal positions by using stereoscopic rendering techniques so that the observer can see a merged image with binocular parallax that appears truly 3D on stereoscopic displays. Compared to the conventional display mode employing single-image cues, stereoscopy provides unambiguous depth information with greater robustness [9], [16].

However, stereoscopic rendering is limited by slow rendering rates when compared to those of single image rendering. In the worst case, the rendering time is doubled. Therefore, fast stereoscopic rendering approaches have been proposed which exploit the coherence between the two views so that the second image of the stereo image pair can be generated in a fraction of the time of the first image. Most of the previous work was based on traditional geometric rendering schemes, such as ray tracing, where the second view requires as little as 5 percent of the computation time to fully ray trace the first view [1]. With the increasing interest in volume-rendering techniques, a fast stereoscopic volume-rendering technique for ray casting was proposed by Adelson and Hansen [2] and further accelerated by He and Kaufman [8]. However, both techniques were based on the assumption of parallel projection. Furthermore, none of the previous work is fast enough for interactive navigation. For example, Adelson and Hodges [1] have generated a pair of stereoscopic images in about 20 minutes.

In recent years, efficient image-based rendering (IBR) has emerged. It rapidly generates a novel view from a set of precomputed or preacquired 2D images (called *reference images*) rather than from a 3D model of the scene, typically by applying 3D warping on the reference images with depth information (called *depth images* [4], [25]). To attain a real-time performance, McMillan and Bishop [17], [18] proposed a fast warping approach that uses an incremental evaluation of the 3D warping equations and an occlusion-compatible ordering algorithm without the expense of Z-buffering. However, the warping from a single depth image results in gaps due to visibility changes. Researchers therefore proposed to use multiple reference images to fill the gaps [4], [14], but the rendering cost increased accordingly. To alleviate this, Max [15] and Shade et al. [25] introduced a layered depth image (LDI) representation, which contains multiple depth pixels at each discrete location in the reference image. Because the LDI data are represented in a single image coordinate system, McMillan's warp-ordering algorithm can be successfully applied.

There is a close relationship between IBR and stereoscopic rendering. For example, both speed up the rendering process by exploiting the coherence between different views and both generate an output image from the available image(s). However, in an interactive navigation system, the reference images in IBR are normally acquired during a preprocessing stage, whereas the left-eye image (in short, left image) in stereoscopic rendering is usually generated on the fly. In addition, as a more general technology, IBR may use more than one reference image to generate the output image. Even if only a single reference image is used [17], [18], the output view can be located at an arbitrary place nearby rather than always horizontally to the right of the reference image. On the other hand, in stereoscopic rendering, although the input is a single image from the

- *M. Wan is with The Boeing Company, PO Box 3707, MC 7L-40, Seattle, WA 98124-2207. E-mail: ming.wan@boeing.com.*
- *N. Zhang, H. Qu, and A.E. Kaufman are with the Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400.*
  E-mail: {nanzhang, huamin, ari}@cs.sunysb.edu.

Manuscript received 3 July 2001; revised 28 Feb. 2002; accepted 19 July 2002. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 114462.

1077-2626/04/$17.00 © 2004 IEEE Published by the IEEE Computer Society

left eye, the 3D model of the scene is usually available for more information. How to accelerate stereoscopic rendering by taking advantage of both IBR techniques and the specific features in stereoscopic projection is the focus of our work.

In this paper, we propose an interactive image-based stereoscopic rendering approach, where most of the pixel values of the right image are derived from the left image by a fast *3D warping* based on specific stereoscopic projection geometry. The gaps in the warped image are then filled by casting rays through those pixels in the gap regions (in short, gap pixels or hole pixels). Our 3D warping includes reprojection, a clipping test, hidden-pixel removal, and splatting. Specifically, we adopt a simplified stereoscopic perspective-projection geometry so that the reprojection can be done in a scanline order. Clipping test and hidden-pixel removal strategies are used during the reprojection to remove those reprojected pixels that are invisible in the right image, whereas the viable ones are splatted onto the right image in a front-to-back order. The kernel sizes of the splats are limited and their footprints are supersampled for antialiasing. Although our front-to-back order is different from McMillan's back-to-front method, it preserves the same correct *alpha blending* during splatting without explicit sorting. More importantly, by using our hidden-pixel removal scheme and the subsequent ray casting for hole filling, our method successfully solves the *exposure error* in McMillan's 3D warping algorithm [18].

We have applied our stereoscopic rendering algorithm in two case studies of virtual navigation systems: virtual flythrough over a terrain and 3D virtual colonoscopy. The first case study is an interactive flythrough over a voxel-based terrain [28], where the 3D volumetric terrain model is obtained from a 2D elevation map and a corresponding color aerial or satellite photograph. A single terrain image (used as the left image) is generated by taking a sequence of equidistant resamplings along the ray cast from each pixel until the ray hits the terrain surface or exits the volume. If a 2D color photograph is available as the terrain texture, we calculate the terrain color at the hit point by using bilinear interpolation; otherwise, we apply a local shading model. The second case study is our interactive virtual colonoscopy [10], [26]. It takes a spiral CT scan of the patient's abdomen and then a 3D voxel-based model of the colon is segmented from the acquired CT data set, which is examined during interactive endoscopic navigation searching for polyps.

The paper is organized as follows: A description of our serial algorithm and its parallel version are given in Sections 2 and 3, respectively, combined with its application in the two case studies. Section 4 discusses how to add view-dependent shading and transparency effects to the stereoscopic rendering. Performance results are reported in Section 5. A detailed comparison between our method and McMillan's 3D warping approach [18] is given in Section 6. An early version of our work was applied to terrain rendering [29], which provided a solution to specific height-field applications without taking advantage of IBR techniques.

2 Stereoscopic Rendering Algorithm

The basic steps of our serial image-based stereoscopic rendering pipeline are as follows:

1. Generate the left image with depth information on the fly.
2. Establish simplified perspective-stereoscopic-projection geometry so that all the pixels in a scanline of the left image are reprojected to the same scanline in the right image.
3. Reproject the left image pixels to the right image in a scanline order.
4. Perform a clipping test and hidden-pixel removal to delete invisible reprojections.
5. Splat each viable pixel onto the right image for reconstruction and antialiasing.
6. Fill the holes (the warping gaps) in the right image by ray casting, which can be accelerated by various application-oriented optimizations.

2.1 Generating the Left Image

Fast rendering of the left image is not the core of a stereoscopic rendering algorithm, but it is critical to the interactive navigation performance. We propose using a high-quality backward ray tracer, which was simplified to volumetric ray casting [13] when applied in volumetric environments. After more than a decade of effort, ray casting has been optimized to be one of the fastest volume-rendering approaches [21], [27].

Note that modern textured polygon rendering also provides high rendering speed and good image quality. However, volumetric ray casting has shown several important advantages in the stereoscopic rendering of voxel-based objects. First, volume ray casting is a direct volume-rendering method which directly renders object voxels, while polygon rendering has an extra step to explicitly extract object surfaces from their voxel-based representation, which is at least inconvenient, if not impossible. Second, ray casting provides an efficient approach to fill randomly distributed hole pixels in the warped image. Third, although forward polygon projection also provides depth information at the cost of reading the Z-buffer, the depths in Z-buffer are not as accurate as those generated during ray casting because the formers are nonlinear. The further the sample point to the near plane of the Z-buffer, the less precise its depth is.

Although we can employ any available high-performance ray casting in our stereoscopic rendering, we prefer to use those ray-casting optimizations that do not sacrifice image quality. Therefore, the *space leaping* optimization [5], which skips over empty voxels that have no contribution to the final rendered image, becomes an attractive method. In this section, we demonstrate two efficient ray-casting optimizations used in our two case studies by exploiting space leaping in their specific scene geometries. Those optimizations will also be used later to accelerate hole filling in the right image. We would like to point out that the special geometries in our two case studies are exploited only for more efficient space leaping during ray casting (which could be easily replaced by a generic space-leaping

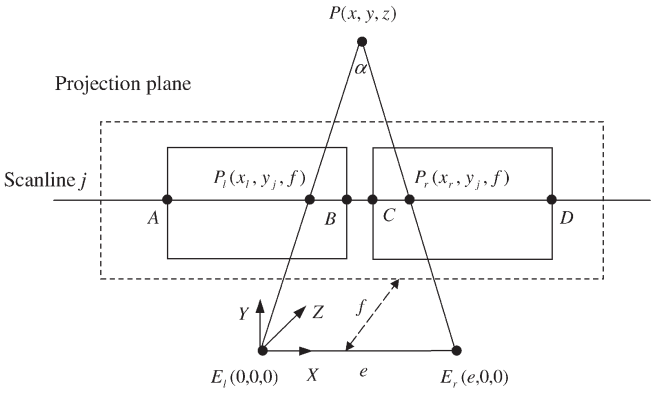


Fig. 1. Stereoscopic perspective projection geometry.

method [5] for more general geometries). Consequently, our stereoscopic rendering algorithm presented in this paper is generally applicable to any geometry in a volumetric scene.

In the virtual flythrough over terrain, we use a ray casting procedure that can be accelerated dramatically by exploiting the special ray coherence in a terrain scene [6], [12], [28], where a higher ray always hits the terrain at a greater distance than that of the ray below it. Therefore, when the image columns are vertical to the horizontal direction, we can accelerate the traversal of a higher ray by space leaping according to the intersection information (or depth value) of the previous ray that emanated from a lower neighboring pixel on the same image column.

In the 3D virtual colonoscopy, we implemented a fast ray casting by exploiting the specific features of the human colon [26]. Note that the human colon has a cavity structure with a bounding thin surface and, during navigation, the camera is always located inside the empty colonic interior. Therefore, we skipped the empty space along each ray by sampling the Euclidean distance field from each voxel inside the colon to the closest colon wall and performed sampling only in the area near the colon surface.

2.2 Stereoscopic Projection Geometry

The fundamental technique of stereoscopic rendering is the establishment of *correspondence*, that is, the pairing up of pixels in the two images such that each pixel in a pair of points is the image of the same object point in space. In our algorithm, we are more interested in finding the corresponding right-image pixel for each *nonbackground* (or nonempty) left-image pixel through which a ray intersects an object surface. To simplify this problem, we adopt the assumption (cf. [19]) that the two image planes (the left-eye and right-eye image planes) are chosen to be coplanar so that each pixel in a scanline of the left image is reprojected to the same scanline in the right image. Usually, since the two eyes are closer to each other than to the objects in space, the above rectification of image planes works fine.

Fig. 1 illustrates the perspective projection geometry used in our stereoscopic rendering algorithm. The left and right centers of projection, named E_l and E_r, separated by a distance e, are placed at the same side of the projection plane with the same distance f, where e and f have been magnified for legibility. We define a lefthand image space coordinate system as follows: Origin O is located at E_l. Axis Z points perpendicularly to the image plane. The main view vector or boresight of the left view is along axis Z. Axis X is along the horizontal direction pointing right, passing through E_l and E_r. Axis Y is orthogonal to both axes X and Z and points upward. In fact, any arbitrary single coordinate system can be specified here, such as placing the two eyes symmetrically around the origin [1]. The stereo images generated from different projection geometries would be very similar because the angle α between a pair of left and right rays is very small (often less than 2 degrees [8]). The advantage of selecting our stereoscopic projection geometry to be the same as our single-image-rendering geometry is that we can use our fast ray casting to generate the left image without any modification.

2.3 Reprojecting Left-Image Pixels

We assume that every pixel in the left image is a perspective projection of an opaque object point in the scene. (In Section 4, we discuss a solution for translucent objects). As a result, a left-image pixel may or may not have a correspondence in the right image, depending on whether the associated scene object point is visible to the right eye. If a left-image pixel does have a reprojection on the right image, the reprojection must be located in the same scanline in the right image, according to our stereoscopic projection geometry. Therefore, our reprojection procedure is performed in a scanline order. Furthermore, the reprojections of all left-image nonbackground pixels in the same scanline are conducted in a common epipolar plane that passes through the two eyes and the scanline. Our reprojection computation thereby only involves x and z coordinates and we only care about the x coordinate of the reprojection for the right view.

Assume that the current pixel in question is at position (i_l, j) of the left image and its 3D location in the image space is at $P_l(x_l, y_j, f)$ on the projection plane (see Fig. 1). We are looking for the corresponding pixel in the right image (i_r, j) of the same image scanline j, whose 3D image space coordinates are $P_r(x_r, y_j, f)$. Assume that these two pixels are the images of an object point $P(x, y, z)$ in the image space, where x, y, z values can be obtained from the depth value of d_l in the left image.[1] Line segments AB and CD are the parts of the projection scanline, respectively, covered by the left and right images. P_l and P_r, respectively, fall into AB and CD. When the two eyes are close enough, AB and CD can overlap.

From $(x_r - e)/(x - e) = f/z$, we get

$$x_r = e + f(x - e)/z. \tag{1}$$

Then, the unknown i_r of the reprojected pixel position (i_r, j) is computed as:

$$i_r = (x_r - c)/w = (e + f(x - e)/z - c)/w, \tag{2}$$

where c is the x-coordinate of the left-most pixel of the right image on the current scanline j and w is the physical width

1. To save computation time, in our implementation, we use an extra 2D array to save the x, y, z coordinates of the sampled surface points during the ray casting of the left image.

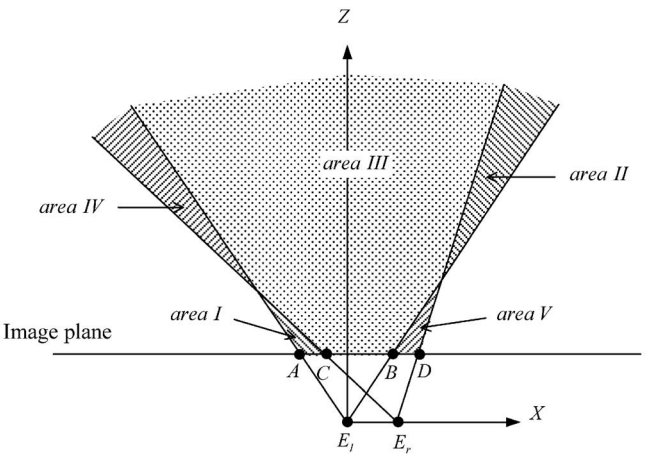


Fig. 2. Different fields of a view from the two eyes.

of one pixel. Since c and w are constants for each scanline, (2) simplifies to

$$i_r = a + b(x - e)/z, \tag{3}$$

where $a = (e - c)/w$ and $b = f/w$ are constants. Therefore, the reprojection of a point from the left image to the right image can be computed using *two* additions and *two* multiplications.

2.4 Resolving Visibility

An object point visible to the left eye may not be visible to the right eye for two reasons. First, the two eyes have different fields of view. Second, object occlusion is viewpoint dependent. In this section, we focus on a viable reprojection determination for left-image pixels in a *front-to-back* order by sequentially performing a clipping test and hidden-pixel removal. Fig. 2 illustrates the different fields of a view from the two eyes. Scene areas *I* and *II* are only visible to the left eye. Areas *IV* and *V* are only visible to the right eye. Area *III* is visible to both eyes. Note that the small area from the image plane toward the eyes is excluded from the fields of a view.

We perform a clipping test on each left-image pixel P_l as follows: If P_l is reprojected beyond the scope of $[C, D]$ on the current scanline, the related object point P in space must be located in either *I* or *II* and invisible from the right eye. So, it is rejected. Otherwise, the reprojection of P_l is located between C and D and its related object point P is in area *III*. Although P is now in the field of view of the right eye, it may still not be viable because scene objects located in front of P from the right view can occlude it. Because these occluding objects may be located either in area *III* or *V*, we further use two different detection strategies. If the occluding objects appear in area *V*, they are newly exposed or new-incoming objects in the right image. So, we cast rays through the right-image pixels between B and D on the scanline to detect these objects. Because the hitting test along these rays is performed only in the small area *V*, this can be done very fast. If the occluding objects are located in area *III*, we employ an effective *hidden-pixel removal* to remove all potentially occluded reprojections. Specifically, from the relative positions between the two eyes, we observe that the reprojection of a left-image pixel P can only be occluded by the reprojections of those left-image pixels, say Q, located to the right of P on the same image scanline, no matter whether pixels P and Q belong to the same surface or not. Accordingly, we conduct the following hidden-pixel removal during the reprojection of each pixel in the current left-image scanline in a right-to-left order, which guarantees that reprojected pixels arrive at the right image in a front-to-back order.

Assume that P_l is the current pixel in the left image, P_r is P_l's reprojection to the right eye, P_l and P_r are the images of an object point P located in area *III*, and P_s is the current left-most reprojection of those left-image pixels located to the right of P_l on the same scanline (see Fig. 3). Our hidden pixel removal determines whether P is to be occluded in the right image and, therefore, P_r is to be rejected, according to the relative positions between P_r and P_s. Obviously, P_r is either to the left of P_s or not. If P_r is to the left of P_s, then P is visible to the right eye (see Fig. 3a). Therefore, P_r is viable and we update P_s by P_r. Otherwise, P_r is located at or to the right of P_s, which means P could be occluded by object surfaces that are sampled by the left-image pixels to the right of P_l. In this case, we reject P_r and keep the current P_s untouched, no matter whether it is actually occluded (as in

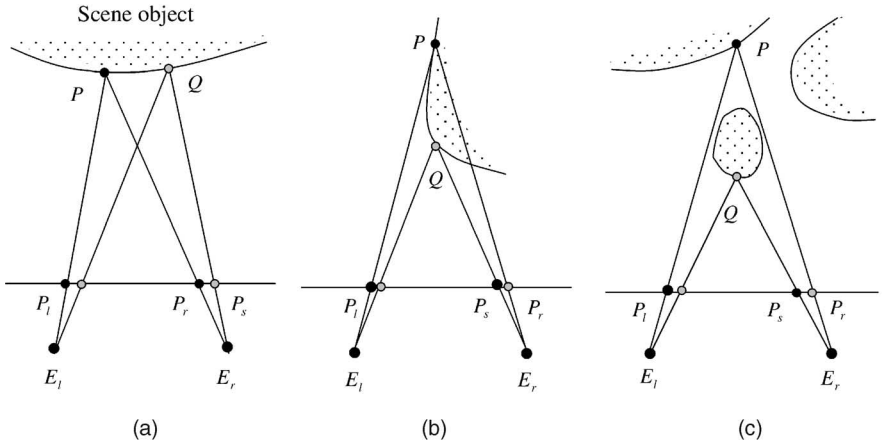


Fig. 3. Hidden-pixel removal.

the case of Fig. 3b) or not (as in the case of Fig. 3c) in the right image. Such a uniformed conservative rejection has two important advantages. First, it simplifies the implementation of the algorithm. Second, it avoids the exposure error in [18] because it removes the possible source (i.e., point P_r when it appears to the right of P_s) for the error. The subsequent hole filling will take care of the possible gaps caused by the rejection of actually viable points so that our hidden-pixel removal scheme will not cause artifacts in the final image. Hole filling will also take care of those object points newly exposed at area *IV* that are visible through the gaps in the right image.

2.5 Reconstruction and Resampling

Since the left image is represented by a 2D array of discrete samples, reconstruction and resampling are needed to synthesize the right image. According to the above reprojection sequence, pixels are drawn in the right image in a front-to-back order along each scanline without explicit depth sorting. This makes splatting an efficient solution to the reconstruction and resampling problems.

The splat size can be computed from the following formula [25]:

$$size = \frac{(d_1)^2 \cos\theta_2 res_2 \tan(fov_1/2)}{(d_2)^2 \cos\theta_1 res_1 \tan(fov_2/2)}, \quad (4)$$

where d_i is the distance from the sampled surface point to eye i, θ_i is the angle between the surface normal and the ray from eye i to the surface point, res_i and fov_i are related to the resolution and field of view of eyes i, respectively. Since res_i and fov_i do not change from eye to eye in our algorithm, we simplify the above equation as:

$$size = \frac{(d_1)^2 \cos\theta_2}{(d_2)^2 \cos\theta_1}. \quad (5)$$

By further applying the approximation principles of [25] to our simplified geometrical situation of a stereo pair, we conclude that a splat size of one pixel is appropriate. But, in our implementation, we use fixed-size splats of two-pixel width, which is larger than the one-pixel width, to cover most of the small gaps on the warped image. However, we still need to treat some right image pixels as holes if the splatting weights at these pixels are below a threshold (which means these parts are stretched too much). We later update these pixels by ray casting.

For antialiasing, the footprints of the splats are supersampled in the right image. This is implemented by reconstructing a supersampled scanline for splatting and then filtering it back to the desired resolution. Therefore, each pixel in a footprint is split into eight subpixels, where each subpixel has an alpha value to approximate the Gaussian splat kernel. This supersampling representation of a footprint is derived from the A-buffer technique [3], [15], [29]. However, it results in a more accurate reconstruction by a Gaussian filter rather than a linear interpolation using the A-buffer technique. It is important to point out that the footprints of our splats have only one dimension along each image scanline. Therefore, our 1D splatting with supersampling is much faster than normal 2D splatting.

In fact, similar concepts to our hidden pixel removal and splatting have been used in IBR algorithms. For example, "surfels" [22] uses splatting for both image reconstruction and sample point visibility detection. If two sample points have a depth difference of less than a threshold, they are treated as points on the same surface; otherwise, Z-buffer is used to resolve the visibility. Since a surfel conducts a sufficient sampling in the object space, it only produces small holes in the warped image, which can be simply filled by interpolations between neighboring splats. "WarpEngine" [24] used a different method to detect depth discontinuities based on the surface curvature. If the second derivative of a reference image sample exceeds a threshold, the sample is marked as disconnected. Splatting is also used for translucent objects in "surface splatting" algorithm [31]. Surface splatting divides the Z-buffer into buckets and then splats samples into different buckets based on their Z depths. After blending samples of the same bucket into one image, all images from different buckets are composited from back to front.

Note that our hidden pixel removal is more efficient and simpler than those visibility detection methods used in [22], [24] because it does not detect surface connectivity. Although it may reject some actually visible samples, our hole filling procedure takes care of the possibly "enlarged" holes. In fact, even if we used the continuity detection methods [22], [24], we were unable to avoid large holes in the right image because the left image normally cannot sufficiently sample the volumetric scene. For the same reason, we cannot use surface splatting [31] in our stereoscopic rendering to generate a semitransparent right image (see Section 4 for our solution to translucent rendering).

2.6 Filling Holes in the Right Image

During warping of left-image pixels in the *right-to-left* order along each scanline, three different kinds of gaps may appear, depending on whether the current reprojection P_r of the nonbackground left-image pixel P_l is the right-most, middle, or left-most reprojection on the right-image row. Since background pixels have no depth information, they are not reprojected. We use C and D to represent the left endpoint and right endpoint of the current right-image scanline and assume P_s to be the current left-most reprojection on the current right-image scanline.

- Right-end gap (Fig. 4a): First, we examine whether P_r is the first viable reprojection on the current right-image row by using our clipping test. If it is, we splat P_l around position P_r on the current right-image row, cast rays through hole pixels (if they exist) between P_r and D, whose weights are below a user-defined threshold, to fill this right-end gap, and initialize P_s by P_r.
- Middle gaps (Fig. 4b): If P_r is to the left of the current left-most reprojection P_s, then P_r passes our hidden-pixel removal test. Therefore, we splat P_l around P_r on the right-image row, blend its kernel with previously splatted pixels (if they exist), cast rays through hole pixels (if they exist) between P_r and P_s to fill this middle gap, and update P_s by P_r.

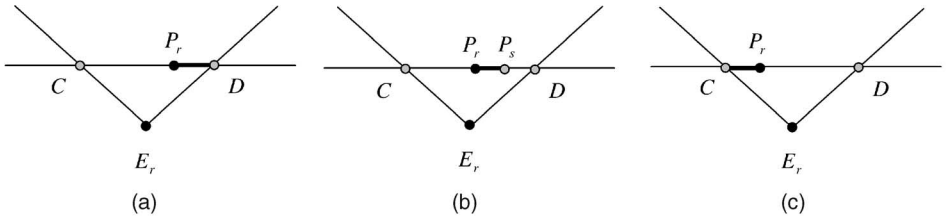


Fig. 4. Mark holes during 3D warping in a scanline order.

- Left-end gap (Fig. 4c): If P_r is the left-most reprojection passing our clipping test, we cast rays through the hole pixels (if they exist) between C and P_r to fill the left-end gap.

The pseudocode in Table 1 summarizes our serial stereoscopic rendering algorithm.[2] It indicates that all those right-image pixels with zero or small weighs from splatting need ray casting, including those background pixels which do not intersect with any surface in the scene. Therefore, accelerating ray casting during hole filling is also critical for interactive performance. Different applications may benefit from different ray-casting optimization strategies. In the following section, we present our implementations of fast ray casting by exploiting different space coherences in the two case studies, incorporated with the design of efficient image-based task partitioning schemes toward efficient parallelization of our serial stereoscopic rendering algorithms.

3 Parallel Stereoscopic Rendering Algorithm

To maintain interactive rates during stereoscopic rendering, we parallelize the generation of the left image as well as the right image. The latter is accomplished by separating our serial stereoscopic rendering algorithm into two passes: a 3D-warping pass (including reprojection, a clipping test, hidden-pixel removal, and splatting) followed by a hole-filling pass (using ray casting). In the first pass, pixels in the left image are warped and splatted into the right image in scanline order. When holes appear in a right-image row (cf. Fig. 4), we simply assign a special color to those hole pixels. When all pixels in the left image have been processed, we enter the hole-filling pass to compute the marked hole pixels using ray casting. There are several advantages with this separation. First, separating warping and hole filling passes provides the best cache coherency because different data are accessed in different passes. Second, since each pass has different computation complexity, the separation enables the design of the most suitable task-partitioning scheme for each pass for parallelism. In addition, the number and distribution of hole pixels in the right image are critical to the design of a load-balancing partition scheme for parallel hole filling. This information varies among different frames. By separating the two passes, this information is available at the end of the first pass. In this section, we demonstrate different image-based task partitioning schemes to effectively parallelize these two passes as well as the generation of the left image in the two case studies.

3.1 Parallelizaton of Left Image Generation

In our flythrough system, we used a static image-based task-partitioning strategy for parallel rendering of single images. The image was treated as a pool of columns and each processor processed a fixed number of image columns in an interleaved order. Ray coherence was exploited to skip empty space during ray casting. Since the total traversal distance along each set of rays cast from each image column was almost the same, each set of rays had approximately an equal amount of work to perform during ray traversal [28].

In the 3D virtual colonoscopy system, we employed a different image-based static task-partitioning scheme to generate a full ray-casting image [26], [30]. Specifically, each image was divided into equal-sized rectangular blocks, such as four by four for 16 processors. Each pixel of the block was allocated to one processor for ray casting using our distance-field-assisted optimization. Because the colon wall is very thin and the empty space inside the colon has been skipped quickly along each ray, the rays in a local image block have a very similar amount of work during ray traversal. Therefore, we achieved a good load balance.

3.2 Parallelization of 3D Warping

In a 3D warping pass, the main computation comes from reprojection and splatting. Since we use fixed-size splats, the amount of computation does not change much at each left-image pixel. Accordingly, the computational cost on each scanline is proportional to the number of pixels in the left-image row. Therefore, we can treat the left image as a pool of image rows and assign a fixed number of image rows to each processor in an interleaved or contiguous order.

Another factor affecting the design of the task-partitioning scheme for parallel 3D warping is the acceleration for the subsequent hole filling. Although we do not perform ray casting right now to fill holes, we may need to do some preparation during 3D warping for fast hole filling, besides assigning a specific color to each hole pixel. According to our experience, for instance, having the depth information of each hole pixel makes the ray casting much faster. A straightforward method to determine the depth of a hole pixel is using linear interpolation on the depths of the two nearest nonhole pixels located on the left and right sides of this hole pixel on the same right-image row. However, there is no guarantee that the interpolated depth of a ray cast

2. For simplicity of description, the pseudocode does not include the occlusion detection of new incoming objects in scene area V, described in Section 2.4.

TABLE 1
Serial Stereoscopic Rendering Algorithm

```
compute end points C and D on the right-image row;
for (each row in the left image) {
    START = FALSE;
    END = FALSE;
    P_s = C - 1; // initialize to an impossible value

    //from right to left
    for (each left-image pixel P_l){
        calculate the reprojection P_r of P_l;

        //discard scene object points in area II
        if (P_r > D) continue;

        //discard scene object points in area I
        if (P_r < C) END = TRUE;

        if (START==FALSE && END==FALSE){ //right-end gap
            splat P_l around P_r in right image;
            fill gap between [P_r, D] by ray casting;
            P_s = P_r;
            START = TRUE;
        }

        if (END==TRUE){ //left-end gap
            if (START== TRUE){
                fill gap between [C, P_s] by ray casting;
                P_s = C;
            }
            break; //end of reprojection on the current row
        }

        if (P_s - P_r > 0) { //middle gaps
            splat P_l around P_r in right image;
            fill gap between [P_r, P_s] by ray casting;
            P_s = P_r;
        }
    }

    if (P_s >= C) fill gap between [C, P_s] by ray casting;
    if (P_s <  C) fill gap between [C, D] by ray casting;
}
```

pixel is correct except when the two neighbors are very close to each other. Unfortunately, a hole pixel is usually allocated between two neighboring reprojections that are so far away that an uncertain gap appears between them. Our solutions for the two case studies are described below.

In order to perform correct space leaping during the ray casting through hole pixels in the flythrough case study, we choose to exploit ray coherence in the image column order described in Section 2.1. Thus, we can skip over most of the empty space along a ray according to the calculated depth of the nearest hole pixel below it. However, how about the depth of the lowest hole pixel P_w in an image column? There are two different situations. If P_w is the bottom-most pixel of the column, we have to traverse along the ray from the beginning as we did in the left view. Otherwise, P_w has a lower nonhole neighbor, say P_n, whose depth can be used to accelerate the ray casting through P_w. That is, the depth of P_n is of particular importance for fast hole filling along this image column.

For implementation, we establish a 1D array for each processor, called a *linear depth buffer*. Each element of this buffer corresponds to one column of the right image, with several components to record the position of the lowest hole pixel P_w, the depth of its lower neighbor P_n (if it exists), and the number of hole pixels in that image column. Note that the computation of the depth of a splatted pixel is needed only when its upper neighbor is a hole pixel. Therefore, we assign image rows to each processor in a *contiguous order* and we constrain the reprojection procedure in a scanline top-down order on each processor so that we can compute depth values only when necessary. Each processor processes approximately an equal number of nonbackground

left-image pixels. After all processors have completed their warping work, the depth buffers are combined into one and only the lowest hole pixel survives for each column. The hole pixels at the same column are summed up.

In our virtual colonoscopy, parallelism of 3D warping is more general and simpler, without the above extra preparation for fast hole filling. This is because the depth information at each hole pixel is obtained independently from the distance field inside the colon (cf., Section 2.1.2), rather than from its lower neighbor as terrain rendering. We simply employ a general task-partitioning scheme by assigning the image rows in an *interleaved* order to processors for load balancing.

3.3 Parallelization of Hole Filling

Due to the randomly scattered distribution of hole pixels in the warped image, our previous static task-partitioning schemes used to generate a single ray-casting image [26], [28] cannot guarantee a good load balance in filling holes. A dynamic partitioning scheme that considers the specific requirements of fast ray casting in each specific application will be a better solution.

In the flythrough case study, when we use ray casting to fill holes, the amount of computation work may differ dramatically among image columns due to their different number of hole pixels. Therefore, we propose a dynamic partitioning scheme as follows: First, a task queue is created from those right-image columns with holes. Then, all the image columns in the queue are sorted in order of decreasing number of hole pixels. After that, each processor takes and processes one column at a time from the head of the task queue. Once a processor completes its column, it takes another one from the head of the current queue until all columns are exhausted.

For each column, hole filling is completed in three steps. First, find the location of the lowest hole pixel and the corresponding depth from the linear depth buffer generated in the previous warping pass. Second, cast a ray from this hole pixel and use the corresponding depth to skip the empty space along the ray. When the hit position is found along the ray, update the depth buffer with the new depth value. Third, move upward along the column to the next hole pixel with the specific color and repeat the second step until there are no more hole pixels left along the column.

In our experimentation, we found that, except for a small number of columns at the right of the right image, most columns have a very small number of hole pixels (see Fig. 6c, where hole pixels are marked in red). Our dynamic task-partitioning scheme benefits from this fact by sorting the task queue. When approaching the end of the parallel hole-filling pass, all processors are dealing with small-size work units (usually columns with a couple of hole pixels). As a result, there is little idle time for those processors that complete their work earlier than others. Consequently, we have reached good load balancing.

In the virtual colonoscopy case study, a static task-partitioning scheme does not work well when we cast rays only through those scattered hole pixels rather than through all the pixels in each image block. For balanced parallel hole filling, we employ another efficient dynamic image-based partitioning scheme proposed by Nieh and Levoy [20] and later used by Lacroute [11] and Parker et al. [21]. Since we do not want adaptive image sampling optimization during rendering, which trades image quality for speed, the partitioning scheme simplifies as follows: First, the current right image (which contains warping gaps) is divided into small regular blocks that form a task queue. Then, each processor takes one image block at a time and fills holes in this block by ray casting until the task queue is empty. Due to the unequal distribution of hole pixels in the right image, we sort all the image blocks in the queue according to the number of hole pixels they contain. Therefore, image blocks containing more hole pixels are processed earlier so that we can minimize the idle time of those processors which complete their work earlier than others.

4 View-Dependent Shading and Transparency Views

During the above stereoscopic rendering, we implicitly assume that a scene point has the same intensity in the two images when we perform 3D warping. This assumption holds when we render a terrain with a color photo as shown in Fig. 7 and Fig. 8 because the intensities of the pixels in the color photo are view independent. Yet, if a view-dependent lighting model is used for rendering, as shown in Fig. 6 and Fig. 9, the same scene point may have different intensities for two eyes. Then, this assumption is no longer strictly correct and, consequently, image error may be increased. A solution is to recalculate the color of the scene point instead of directly obtaining it from the left image. The color can be computed using the lighting model depending on the current ray direction (from the right eye to the scene point) and the surface normal at that scene object point, which is view independent. To speed up the calculation of the lighting model, we could save the normals for all nonbackground pixels in the left image and further establish a look-up table mapping the ray direction and surface normal to a precalculated lighting color.

Another assumption in our algorithm excludes translucent objects from the scene (cf. Section 2.3), such as a translucent glass window in architectural walkthrough. A typical solution in IBR is to precapture a number of images with their centers of projection equally spaced along a semicircle and with the view oriented toward the center of the window (called *portal* [23]). Unfortunately, this is by no means an accurate solution because the rays from the output image are distorted at the glass window. Surface splatting [31] provides a better solution using a layered frame buffer. However, it is not applicable to our stereoscopic rendering, as discussed earlier.

Our solution is to recalculate the accurate right image by an accelerated ray casting because its pixel values cannot be directly obtained from the left image. First, we compute the depths for most of the right-image pixels by reprojecting the left-image pixels so that we can skip over the empty space along the right-image rays according to their depths. The

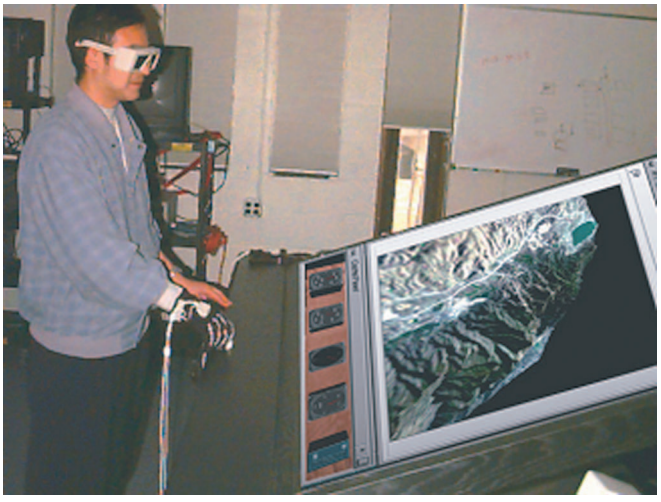


Fig. 5. 3D interactive user interaction on a Responsive Workbench.

depth buffer in our right image is generated by a method similar to but simpler than the surfels visibility splatting [22] because we only need a conservative estimate of the pixel depth. We avoided the expensive scan-conversion in surfels by using the nearest distance from a splat to the image plane as the depth of all the pixels covered by that splat. Second, for the remaining hole pixels whose depth cannot be obtained from splatting, we can exploit a general ray-casting optimization, such as space leaping [5]. In this way, we guarantee an accurate stereoscopic rendering with transparency. Although the rendering speed of the right image may be slower compared to the direct color splatting, it is still faster than a full ray casting of the right image accelerated by the general space leaping [5] because we conduct a single-step jump based on the ray depth instead of multiple-step jumps.

5 Implementation and Experimental Results

We implemented our stereoscopic-rendering algorithm in both case studies on a 16-processor SGI Power Challenge (194 MHz R10000) and a Responsive Workbench. The Workbench immerses the user within the computer-generated virtual environment with a superior 3D interaction (see Fig. 5). Interactive stereoscopic perspective views are displayed on the fly and stereo shutter glasses are used for several people working collaboratively, with immediate visual feedback and high-definition photo-realistic images.

Figs. 6a and 6b show a pair of stereoscopic images of a terrain in Southern California generated by our algorithm. Each image size is 500×400. Our terrain model consists of a 3D terrain volume with a resolution of $512 \times 512 \times 64$ and a corresponding registered aerial photo with a resolution of 512×512. In Fig. 6, instead of mapping the color photo to the terrain, we used a lighting model. Our system provides such a rendering option considering that a color photo may not always be available for terrain data sets. Table 2 presents rendering times for both the left and right images of Figs. 6a and 6b generated by our algorithm on a different number of processors. Near linear scalability has been achieved as the number of processors increased, which was ascribed to our effective task-partitioning schemes.

The speedup ratio between the left and right images was affected by several factors, such as the number of nonbackground pixels in the left image and the number of hole pixels in the warped right image. For an arbitrary view, as shown in Fig. 6, the average time saving of the right image was about 88 percent of the left image for different numbers of processors. In Fig. 6c, we marked with red those pixels in Fig. 6b that were filled by ray casting. The remaining nonbackground pixels were generated by using 3D warping. The ratio between the number of hole pixels and the warped ones was 2.6 percent. However, since the computation for ray casting was much more expensive than that of 3D warping, the hole filling took about one fourth of the entire rendering time for the right image. That is why we believe that speedup for the additional ray casting procedure is important. For comparison purposes, we measured the ray casting time for Fig. 6b without using ray coherence and found that the hole-filling time increased by a factor of four.

To show the accuracy of the right image in Fig. 6b, we rendered a full ray-casting image from the same view (as shown in Fig. 6d) and compared it with Fig. 6b pixel by pixel. The differences were measured as Euclidean distances in RGB $(256 \times 256 \times 256)$ space and displayed in Fig. 6e. The differences are small because we use supersampled splats for antialiasing and ray casting to fill holes. (The intensities shown in the difference map were magnified by a factor of five.)

Fig. 7 gives the same stereoscopic view of the same terrain data set as in Fig. 6. This time we rendered the terrain with texture obtained from the registered premapped color photo. Table 2 also shows the rendering times for both the left and right images in Fig. 7 generated by our algorithm with different number of processors. The average time saving was about 65 percent, which was less than the saving of 88 percent when we rendered the image pair in Fig. 6 with shading rather than texture mapping. This was because bilinear interpolation for texture mapping was less time consuming than the shading computation. Although the time saved for the right image decreased, the total rendering speed for the stereoscopic image pair increased. As a result, we reached a perspective stereoscopic rendering rate at about 10 Hz on 16 processors.

Fig. 8 shows a pair of stereoscopic images generated by our algorithm from a Los Angeles coast data set (image size is 480×340). The terrain model consists of a 3D terrain volume with a resolution of $468 \times 693 \times 64$ and a corresponding registered aerial photo of size 468×693. To create a more realistic environment, we replaced the black background with an image of a blue sky with white clouds. The time saved on rendering the right image was as high as 84 percent and we reached more than 10 Hz interactive stereoscopic rendering rates.

Similar performance has also been shown on CT-scanned colon data sets of real patients. Fig. 9 shows a pair of stereoscopic semitransparent images during the interactive

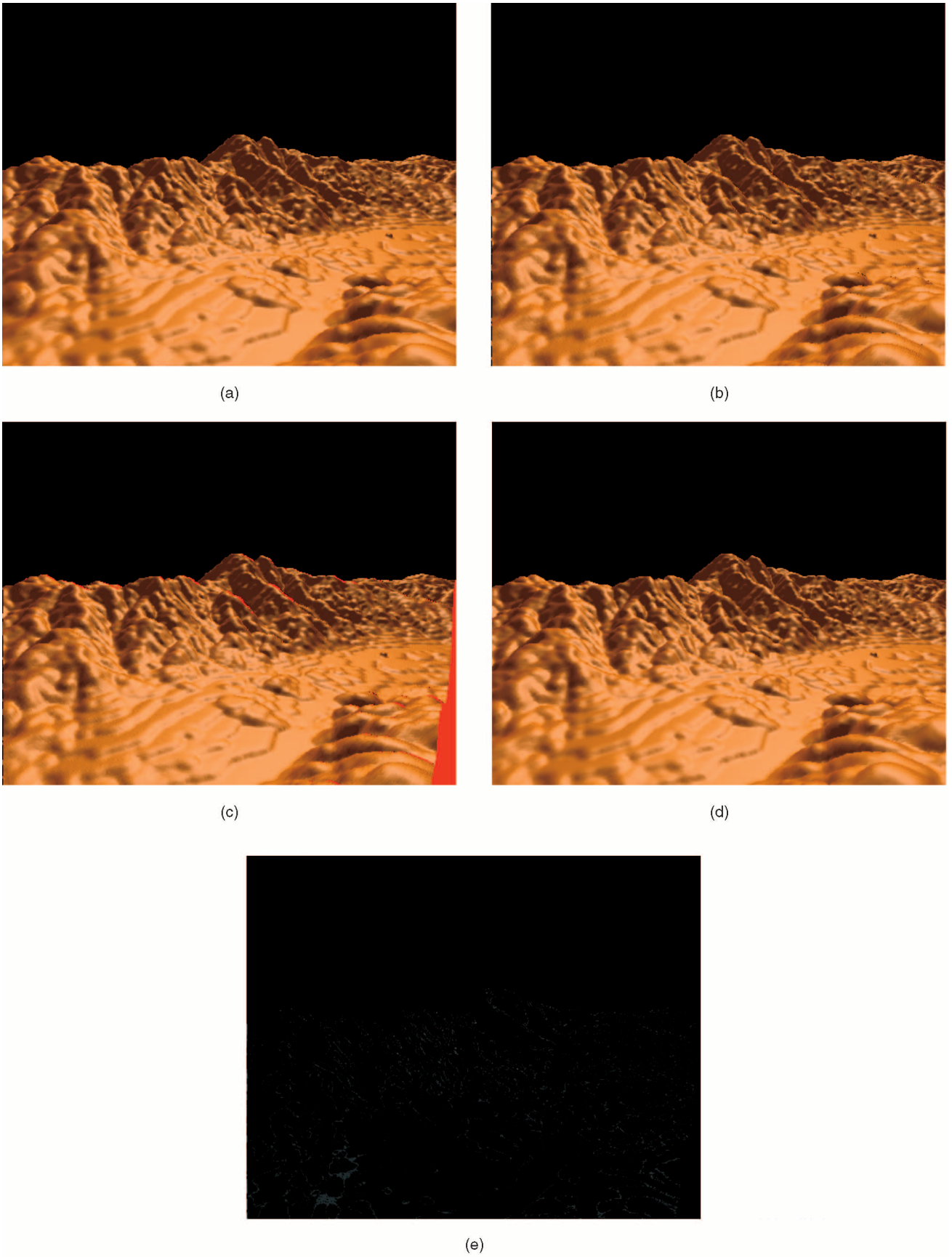
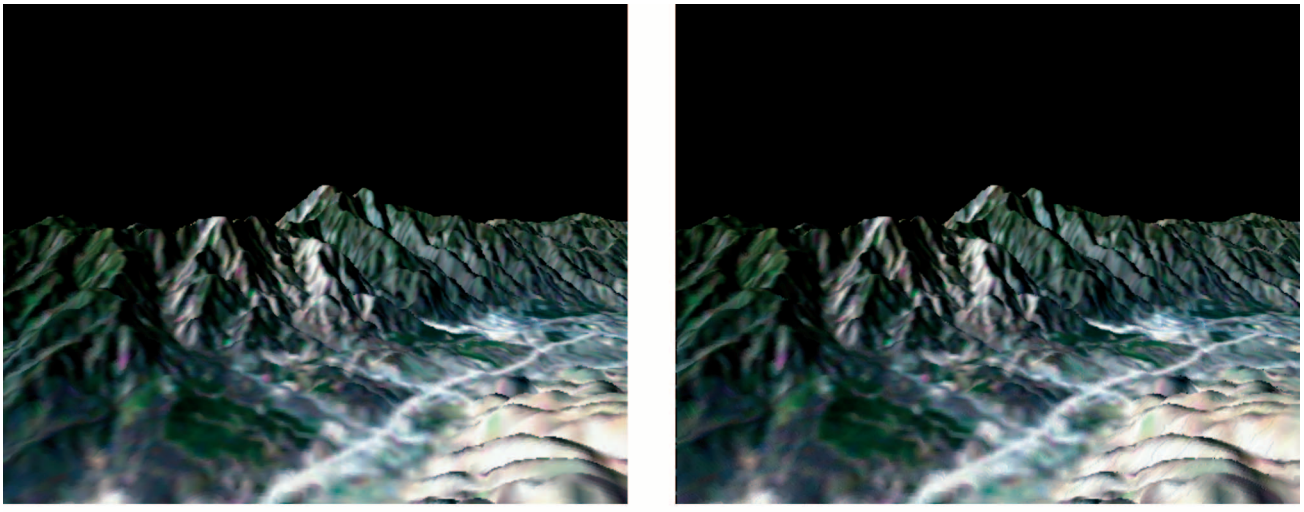
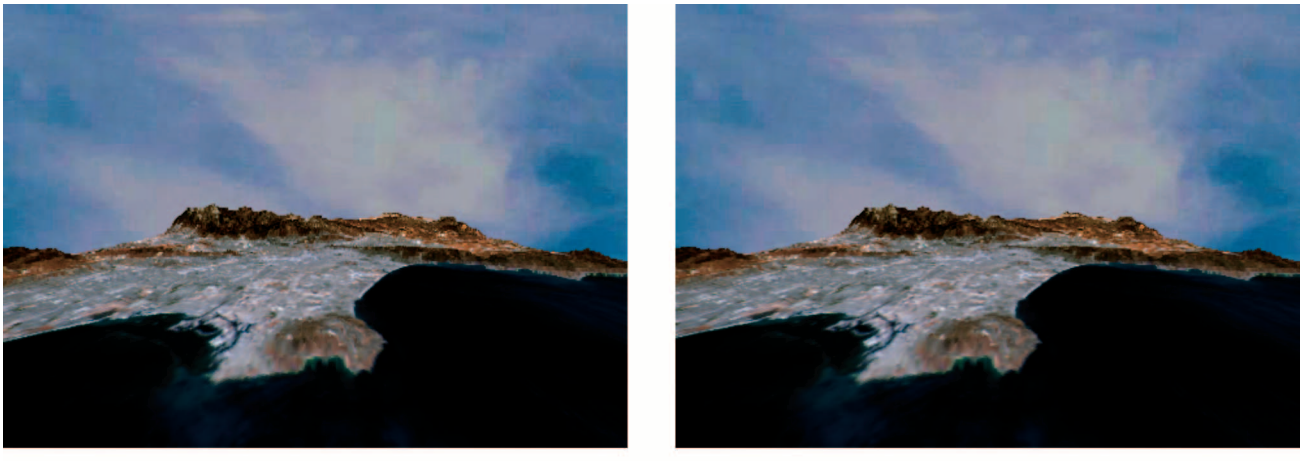


Fig. 6. Stereoscopic rendering of a shaded Southern California terrain on the Workbench. (a) Left image generated by full ray casting. (b) Right image generated by using our method. (c) Hole pixels marked in red. (d) Right image generated by ray casting. (e) Difference map between (b) and (d).



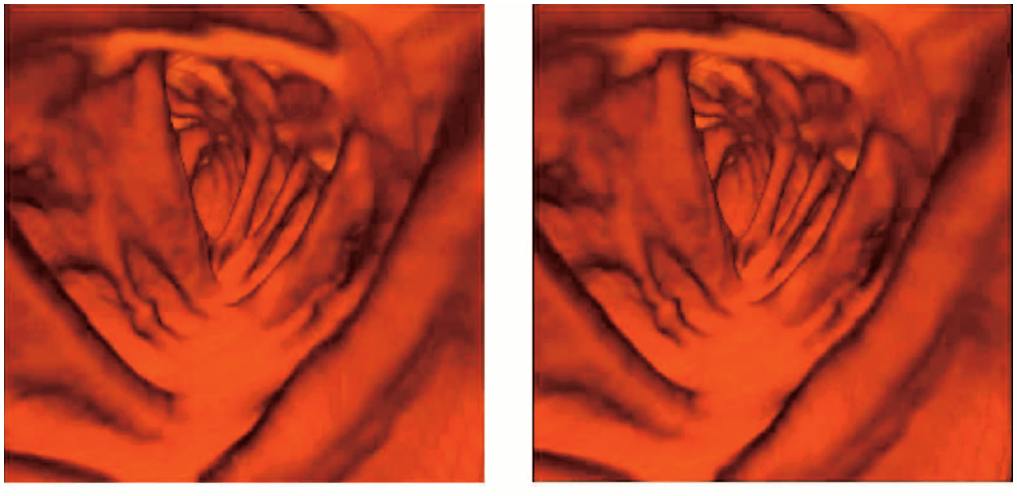
(a) (b)

Fig. 7. A pair of stereo images of a texture-mapped volumetric terrain in Southern California. (a) Left image generated by full ray casting. (b) Right image generated by using our method.



(a) (b)

Fig. 8. A pair of stereoscopic images of texture-mapped volumetric Los Angeles coast. (a) Left image generated by full ray casting. (b) Right image generated by using our method.



(a) (b)

Fig. 9. A pair of stereoscopic images of a patient's colon. (a) Left image generated by full ray casting. (b) Right image generated by using our method.

TABLE 2
Stereoscopic Rendering Times (in *Sec*) of a Terrain in Southern California

Processors:	1	4	8	12	16
Left image (Fig. 6a)	3.21	0.85	0.44	0.33	0.23
Right image (Fig. 6b)	0.38	0.10	0.06	0.04	0.03
Left image (Fig. 7a)	0.97	0.27	0.14	0.11	0.07
Right image (Fig. 7b)	0.32	0.09	0.05	0.04	0.03

navigation inside a $512 \times 512 \times 411$ volumetric human colon. In order to generate the correct compositing and lighting effects in the right image, we "splat" only the depth from the left image rather than the colors of left-image pixels, as discussed in Section 4 for translucent views. In these colon images, we used a high opacity of 95 percent to simulate the nature colon lumen. The rendering times of the left and right images are, respectively, 0.08 sec and 0.03 sec on 16 processors.

6 Comparison with McMillan's 3D Warping

As there are many similarities between our stereoscopic-rendering algorithm and McMillan's 3D warping algorithm [18], the comparison between these two is helpful in evaluating ours. First, both methods generate the output image from one single reference image. Since the reference images are precaptured by McMillan, generating such a reference image database is both time and space consuming. On the contrary, our method is more efficient by generating the left image on the fly, providing the 3D scene model is available and can be rendered at interactive rates.

Second, both methods conduct efficient 3D warping whose cost is approximately determined by the number of pixels in the reference image rather than by the 3D scene complexity as in traditional rendering. When pixels in the reference image are processed sequentially by McMillan, the amount of computation is *six* additions and *five* multiplications. In our method, by taking advantage of the specific stereoscopic projection geometry, we introduce a more efficient warping equation (3) with only *two* additions and *two* multiplications.

Third, both methods resolve visibility without the expense of Z-buffering and, therefore, both allow proper alpha blending during splatting without explicit depth sorting. McMillan proposed spliting the reference image into one, two, or four sheets, depending on the projected position of the output camera in the reference image plane. The pixels in each sheet are then processed in a different scanline order, which guarantees a back-to-front painting of the output pixels. Although using such a painter's algorithm can resolve occlusions correctly, it is less efficient because all the pixels are processed, including the occluded ones. We resolve visibility by scanning the left image in an *opposite* scanline order so that left image pixels are drawn onto the right image in a front-to-back order rather than back to front. As a result, occluded left-image pixels are removed before splatting. In addition, we conduct a simple but effective clipping test in our visibility algorithm and further handle the occlusion from the new incoming objects that are only visible to the right eye (see Section 2.4). These solutions are not covered by McMillan, although he already noticed that ignoring the new incoming objects would cause an *invisible occluder error*.

Fourth, both methods use splatting for image reconstruction. However, McMillan's splatting suffers from two problems. First, as pointed out by Mark [14], the splat approach leaves holes in the output image when using a single reference image. By slightly oversizing the splats, these holes can be reduced or eliminated, but the artifacts associated with oversized splats appear. Second, the splatting reconstruction method causes *exposure errors* when pixels are warped in McMillan's back-to-front order. Exposure errors occur when a background region that should have been occluded is visible in the output image [18]. Our algorithm solves the first problem by limiting the sizes of the splat kernels, supersampling the footprints of these splats, and filling the holes by fast ray casting. To solve the second problem, we perform hidden-pixel removal in a front-to-back reprojection order, which rejects all the potentially occluded left-image pixels and hence removes the source for the exposure errors (see Section 2.4 for more details).

Fifth, both methods are effectively parallelized on shared-memory multiprocessors. Popescu et al. [23] proposed spliting each sheet into P fragments of equal area along epipolar lines, where P is the number of processors. Then, each processor processes pixels in one fragment in McMillan's warping order, except for those close to the two boundary epipolar lines. In a second pass, the remaining pixels are warped in parallel. In our parallel algorithm, we complete parallel 3D warping with a more efficient *one-pass* scanline-based task partitioning scheme rather than a *two-pass* one because we do not need to split the left image into multiple sheets to resolve visibility. According to our stereoscopic projection geometry, the projection of the right eye onto the left image (reference image) is at infinity in the image plane. Therefore, all epipolar lines are parallel to each other, so we can warp the left image pixels along each scanline in the same order. Moreover, the specific feature of a single image sheet in our stereoscopic rendering provides much more flexibility to design effective dynamic image-partitioning schemes which supports fast ray casting through randomly distributed hole pixels.

7 Conclusions and Future Work

We have presented a fast stereoscopic-rendering algorithm for volumetric environments, providing a volumetric scene

model and a fast ray-casting algorithm are available. Exploiting the frame coherence between the two views vastly accelerates the generation of the right image of the stereo image pair. Most of its pixel values are directly obtained from the left image by a fast 3D warping based on a specific stereoscopic projection geometry. A small number of rays are quickly cast through the remaining pixels to fill in holes, by employing ray-casting optimizations. High image quality is ascribed to both the accurate ray casting and our 3D warping with supersampled footprints of splats. Our algorithm has shown good speedups on a multiprocessor by employing load-balancing task-partitioning schemes.

Although our current algorithm has reached interactive rates, we are exploring other antialiasing resampling techniques that are faster than the 1D splatting method we used. One attractive candidate is Fant's nonaliasing 1D resampling interpolation technique [7], which provides a fast mapping from discrete input pixels to discrete output pixels. Such a 1D interpolation method sounds particularly appropriate for our algorithm because the resampling in the right image is exactly performed in a 1D image-row order. However, we have to adapt Fant's method because it works on even-spaced contiguous input pixels only, while our reprojected pixels are unevenly located on each right-image row. One possible solution would be using nonconstant scaling factors at reprojected pixels determined by the varied spacing of the pixels.

Acknowledgments

This work has been partially supported by US Office of Naval Research grant N000149710402, US National Science Foundation grant IIS0097646, and US National Institutes of Health grant CA79180. The patients' data sets were provided by the University Hospital of the State University of New York at Stony Brook. The authors would like to thank the reviewers for their constructive comments, and Aamir Sadiq, Qingyu Tang, Suya You, Milos Sramek, and other team members in the virtual flythrough and virtual colonoscopy projects at SUNY Stony Brook.

References

[1] S. Adelson and L. Hodges, "Stereoscopic Ray-Tracing," *The Visual Computer,* vol. 10, no. 3, pp. 127-144, Dec. 1993.
[2] S. Adelson and C. Hansen, "Fast Stereoscopic Images with Ray-Traced Volume Rendering," *Proc. 1994 Proc. Symp. Volume Visualization,* pp. 3-9, Oct. 1994.
[3] L. Carpenter, "The A-Buffer, an Antialiased Hidden-Surface Method," *Proc. SIGGRAPH,* pp. 103-108, July 1984.
[4] S. Chen and L. Williams, "View Interpolation for Image Synthesis," *Proc. SIGGRAPH '93,* pp. 279-288, 1993.
[5] D. Cohen and Z. Sheffer, "Proximity Clouds—An Acceleration Technique for 3D Grid Traversal," *The Visual Computer,* vol. 10, no. 11, pp. 27-38, 1994.
[6] D. Cohen-Or, E. Rich, U. Lerner, and V. Shenkar, "A Real-Time Photo-Realistic Visual Flythrough," *IEEE Trans. Visualization and Computer Graphics,* vol. 2, no. 3, pp. 255-265, Sept. 1996.
[7] K. Fant, "A Nonaliasing, Real-Time Spatial Transform Technique," *IEEE Computer Graphics and Applications,* vol. 6, no. 1, pp. 71-80, Jan. 1986.
[8] T. He and A. Kaufman, "Fast Stereo Volume Rendering," *Proc. IEEE Visualization Conf.,* pp. 49-56, Oct. 1996.
[9] L. Hodges, "Tutorial: Time-Multiplexed Stereoscopic Computer Graphics," *IEEE Computer Graphics and Applications,* vol. 12, no. 2, pp. 20-30, Mar. 1992.
[10] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He, "Virtual Voyage: Interactive Navigation in the Human Colon," *Proc. SIGGRAPH,* pp. 27-34, 1997.
[11] P. Lacroute, "Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization," *Proc. Parallel Rendering Symp.,* pp. 15-22, 1995.
[12] C. Lee and Y. Shin, "An Efficient Ray-Tracing Method for Terrain Rendering," *Proc. Pacific Graphics,* pp. 181-193, 1995.
[13] M. Levoy, "Display of Surface from Volume Data," *IEEE Computer Graphics and Applications,* vol. 8, no. 5, pp. 29-37, May 1988.
[14] W. Mark, "Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping," PhD dissertation (also UNC Technical Report TR99-022), Univ. of North Carolina at Chapel Hill, 1999.
[15] N. Max, "Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers," *Proc. Eurographics Rendering Workshop,* pp. 165-174, 1996.
[16] *Stereo Computer Graphics and Other True 3D Technologies.* D. McAllister, ed., Princeton Univ. Press, 1993.
[17] L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. SIGGRAPH,* pp. 39-46, 1995.
[18] L. McMillan, "An Image-Based Approach to Three-Dimensional Computer Graphics," PhD dissertation (also UNC Technical Report 97-013), Univ. of North Carolina at Chapel Hill, 1997.
[19] V. Nalwa, *A Guide Tour of Computer Vision.* Addison-Wesley, 1993.
[20] J. Nieh and M. Levoy, "Volume Rendering on Scalable Shared-Memory MIMD Architectures," *Proc. 1992 Workshop Volume Visualization,* pp. 17-24, Oct. 1992.
[21] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive Ray Tracing for Isosurface Rendering," *Proc. IEEE Visualization Conf.,* pp. 233-238, Oct. 1998.
[22] H. Pfister, M. Zwicher, J. van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives," *Proc. SIGGRAPH,* pp. 335-342, 2000.
[23] V. Popescu, A. Lastra, D. Aliaga, and M. Neto, "Efficient Warping for Architectural Walkthroughs Using Layered Depth Images," *Proc. IEEE Visualization Conf.,* pp. 211-215, Oct. 1998.
[24] V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, and L. Nyland, "The WarpEngine: An Architecture for the Post-Polygonal Age," *Proc. SIGGRAPH,* pp. 433-442, 2000.
[25] J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered Depth Images," *Proc. SIGGRAPH,* pp. 231-242, 1998.
[26] M. Wan, Q. Tang, A. Kaufman, Z. Liang, and M. Wax, "Volume-Rendering Based Interactive Navigation within Human Colon," *Proc. IEEE Visualization Conf.,* pp. 397-400, Oct. 1999.
[27] M. Wan, A. Kaufman, and S. Bryson, "High Performance Presence-Accelerated Ray Casting," *Proc. IEEE Visualization Conf.,* pp. 379-386, Oct. 1999.
[28] M. Wan, H. Qu, and A. Kaufman, "Virtual Flythrough over a Voxel-Based Terrain," *Proc. IEEE Virtual Reality Conf.,* pp. 53-60, Mar. 1999.
[29] M. Wan, N. Zhang, A. Kaufman, and H. Qu, "Interactive Stereoscopic Rendering of Voxel-Based Terrain," *Proc. IEEE Virtual Reality Conf.,* pp. 197-205, Mar. 2000.
[30] S. You, L. Hong, M. Wan, K. Junyaprasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang, "Interactive Volume Rendering for Virtual Colonoscopy," *Proc. IEEE Visualization Conf.,* pp. 433-436, Oct. 1997.
[31] M. Zwicher, H. Pfister, J. van Baar, and M. Gross, "Surface Splatting," *Proc. SIGGRAPH,* pp. 371-378, July 2001.

Ming Wan received the BS, MS, and PhD degrees in computer science. He is a senior research scientist at Boeing Phantom Works' Mathematics and Computing Technology organization, joining Boeing in 2000. Before that, he was a research assistant professor in the Computer Science Department at the State University of New York at Stony Brook. His research interests include 3D computer graphics, volume visualization, path planning, virtual navigation, haptics, 3D medical imaging, and expert systems. He is currently the principal investigator of Boeing's Haptic Research project and coprincipal investigator of Boeing's Visualization and Interaction project.

Nan Zhang received the BS degree in computer science from Xi'an Jiaotong University, China, in 1992 and the MS degree in computer science from Tsinghua University, China, in 1995. He is a PhD candidate in the Department of Computer Science at the State University of New York at Stony Brook. His current research interests include volume modeling, terrain rendering, and volume rendering.

Huamin Qu received the BS degree in mathematics from Xi'an Jiaotong University, People's Republic of China, in 1988, and the MS degree in computer science from the State University of New York at Stony Brook in 2000. He is a PhD candidate in computer science at the State University of New York at Stony Brook. His research interests include image-based rendering, volume graphics, medical imaging, and virtual reality.

Arie E. Kaufman received the BS degree in mathematics and physics from the Hebrew University of Jerusalem in 1969, the MS degree in computer science from the Weizmann Institute of Science, Rehovot, in 1973, and the PhD degree in computer science from Ben-Gurion University, Israel, in 1977. He is the director of the Center for Visual Computing (CVC), a Leading Professor and Chair of the Computer Science Department, and Leading Professor of Radiology at the State University of New York at Stony Brook. He was the founding editor-in-chief of the *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 1995-1998. He has been the cochair for multiple Eurographics/Siggraph Graphics Hardware Workshops and Volume Graphics Workshops, the papers/program cochair for the ACM Volume Visualization Symposium (1992, 1994, 1998) and for IEEE Visualization (1990-1994), and the cofounder and member of the steering committee of the IEEE Visualization conference series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is an IEEE fellow and the recipient of a 1995 IEEE Outstanding Contribution Award, the 1996 IEEE Computer Society's Golden Core Member, 1998 ACM Service Award, and 1999 IEEE Computer Society's Meritorious Service Award, and 2002 State of New York Entrepreneur Award. He has conducted research and consulted for more than 30 years specializing in volume visualization, graphics architectures, algorithms, and languages, virtual reality, user interfaces, and multimedia. For more information see http://www.cs.sunysb.edu/~ari.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.