Binary-Space-Partitioned Images for Resolving Image-Based Visibility

(final version - regular paper)

Chi-Wing Fu [†] Tien-Tsin Wong [‡] Wai-Shun Tong [*]

cwfu@cs.indiana.edu ttwong@acm.org cstws@cs.ust.hk

Chi-Keung Tang [*] Andrew J. Hanson [†]

cktang@cs.ust.hk hanson@cs.indiana.edu

[†] Indiana University, Bloomington, Indiana 47405, USA

[‡] The Chinese University of Hong Kong

[*] The Hong Kong University of Science and Technology

Chi-Wing Fu

Computer Science Department, Indiana University, Bloomington, Indiana, 47405, USA.

Tel: +1-812-856-5230 Fax: +1-812-855-4829 Email: cwfu@cs.indiana.edu

Tien-Tsin Wong (* Corresponding author)

Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong.

Tel: +852-26098433 Fax: +852-26035024 Email: ttwong@acm.org

Wai-Shun Tong

Department of Computer Science, The Hong Kong University of Science & Technology, Clear Water Bay, Hong Kong.

Email: cstws@cs.ust.hk

Chi-Keung Tang

Department of Computer Science, The Hong Kong University of Science & Technology, Clear Water Bay, Hong Kong.

Tel: +852-2358-8775 Fax: +852-2358-1477 Email: cktang@cs.ust.hk

Andrew J. Hanson

Computer Science Department, Indiana University, Bloomington, Indiana, 47405, USA.

Tel: +1-812-855-5855 Fax: +1-812-855-4829 Email: hanson@cs.indiana.edu

Binary-Space-Partitioned Images for Resolving Image-Based Visibility

Abstract

We propose a novel 2D representation for 3D visibility sorting, the *Binary-Space-Partitioned Image (BSPI)*, to accelerate real-time image-based rendering. BSPI is an efficient 2D realization of a 3D BSP tree, which is commonly used in computer graphics for time-critical visibility sorting. Since the overall structure of a BSP tree is encoded in a BSPI, traversing a BSPI is comparable to traversing the corresponding BSP tree. BSPI performs visibility sorting efficiently and accurately in the 2D image space, by warping the reference image triangle-by-triangle, instead of pixel-by-pixel. Multiple BSPIs can be combined to solve "disocclusion," when an occluded portion of the scene becomes visible at a novel viewpoint. Our method is highly automatic, including the tensor voting preprocessing step that generates candidate image partition lines for BSPIs, filters the noisy input data by rejecting outliers, and interpolates missing information. Our system has been applied to a variety of real data, including stereo, motion, and range images.

Index terms:— Visibility sorting, binary space partitioning, image-based rendering, segmentation.

1 Introduction

Real-time rendering of complex scenes has long been a challenge for computer graphics. Image-based modeling and rendering (IBMR) techniques approach this problem by using image data directly to decouple the rendering time from the scene complexity. Input images acquired from fixed viewpoints are then warped and composited to generate novel views.

The rendering problem then reduces to that of synthesizing acceptable novel views. A typical solution is to warp the image in a pixel-by-pixel manner [13], [31], [51]. A potentially faster way is to exploit image space coherence by triangulating the image and warping the unstructured triangles instead of pixels. We show in this paper that by subdividing the image using 2D binary space partitioning (BSP), we obtain a hierarchical image representation, namely the *binary-space-partitioned image (BSPI)*, that allows us to perform real-time visibility sorting entirely in

the 2D image space.

The BSP representation has long been exploited in graphics systems, especially for game applications [1], due to its efficiency in solving the visibility [22], [21], [42], collision detection, and shadow generation [14], [15] problems. Methods for generating good BSP trees are typically studied as a part of computational geometry (see, e.g., [6], [44], [2], [16], [46], [61], [9]). On the other hand, the BSP representation has also been used to encode two-dimensional images [50], [48], [49] or even volume data [55]. In this paper, we show that the 2D BSPI is an efficient approach to visibility computation for image-based rendering.

The primary contribution of this paper is twofold:

- A hierarchical image-based representation, the *Binary-Space-Partitioned Image (BSPI)*, is proposed to efficiently encode data coherence in image space and to resolve visibility in linear time. The 2D BSPI is actually an embedding of a 3D BSP tree. Hence, visibility sorting can be performed to correctly and efficiently render novel images by traversing the embedded BSP tree. To handle scenes with occlusions, multiple BSPIs can be used.
- By using *epipolar geometry*, we show that visibility sorting can be performed completely in 2D (rather than 3D) without using any 3D information such as depth or disparity. No depth buffering [10] is required.

The secondary contribution consists of automating the whole process, improving the rendering quality, and also the capability of handling real and noisy data obtained from stereo, motion, and sensing devices:

- We make use of *tensor voting* [40] to automatically refine noisy reference images and segment them into a tree structure based on region partition boundaries. Tensor voting determines the edge locations separating coherent pixel groups, thus subdivides the image into triangles rather

than pixels. This is achieved by the robust detection of discontinuity curves [60].

• The BSPI is a unified approach for a variety of image data, including range images, stereo image pairs, and image sequences. To demonstrate its wide applicability and practicability, all examples used in this paper are real rather than synthetic data.

Our work is closely related to the work by Subramanian and Naylor [55] on representing images using a BSP tree. The focus of our work is on resolving visibility in image space.

2 Related Work

2.1 Visibility in Geometry-based Computer Graphics

Hidden-surface removal is a classical problem in computer graphics. Analytic studies of visibility algorithms have been carried out in [37], [41]. Related methods can be roughly subdivided into three categories: image-space, object-space, and hybrid methods.

Image-space methods resolve visibility after projecting objects onto the screen. Depth buffering [10], [29] keeps track of a per-pixel depth value to determine the closest object fragment. It is the most popular approach used in current graphics hardware design. On the other hand, while scanline algorithms [65], [7], [8], [64], [52], [17] exploit scanline coherence, the required data structure is relatively complex.

Object-space algorithms resolve visibility in the 3D object space. The 3D scene is first partitioned and represented by various hierarchical structures such as cells [3], [59], [58], octrees [26], [56], kd-trees [33], or BSP trees [22], [21], [12] in a preprocessing stage. During rendering, the visibility is resolved by traversing the hierarchical structures. A back-to-front ordering can also be generated in some cases [45], [54], [22], [21] and fed to a Painter's algorithm for rendering. Moreover, the hierarchical nature of these representations also facilitates

fast culling of invisible objects in a complex scene in order to achieve interactive frame rates [3], [23], [24], [59], [58].

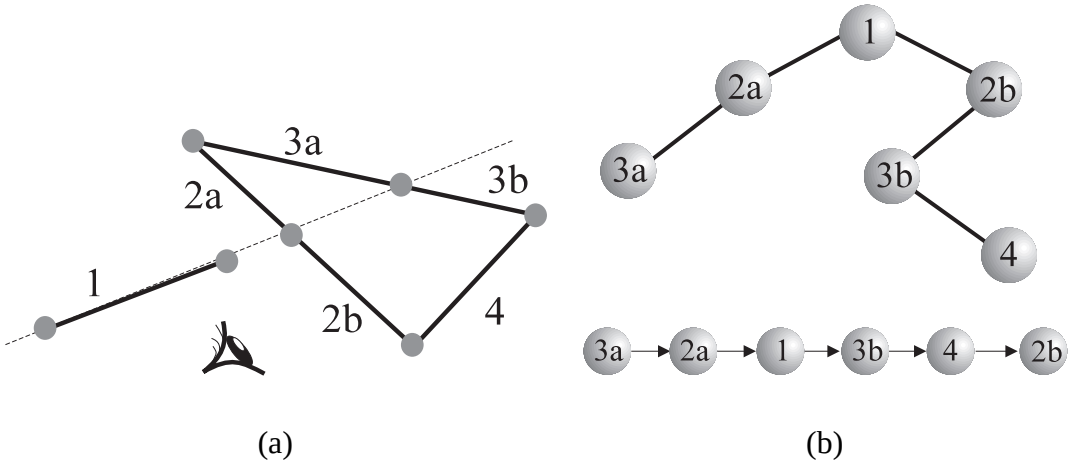


Fig. 1. An example of a binary space partitioning. (a) The scene (top view). (b) The BSP tree yielding the visibility order corresponding to the eye in (a).

Among these structures, the BSP tree is widely used due to its robustness and efficiency. Our proposed BSPI method can be regarded as an image-based version of the classical BSP-tree-based visibility algorithm. For a brief description of BSP trees, consider the example in Fig. 1. In each iteration, a polygon is chosen as the root node of the current binary subtree. It is then extended to form a partition plane (hyperplane) that divides the scene into two half-spaces. The resultant BSP tree is shown in the upper part of Fig. 1b. In general, there is no restriction on the selection of partition planes. The plane need not contain a polygon. To render the view at the eye position in Fig. 1a, a variant of the inorder traversal ($O(n)$ running time where n is number of nodes in the tree) of the BSP tree can be used to generate the drawing order from back to front (Fig. 1b, bottom). Note that this back-to-front drawing order is essential for rendering transparent objects correctly, which depth buffering cannot accommodate. In addition, the algorithm is robust and view independent. Its efficiency explains why it is favored in the

design of real-time computer games [1]. Variants can also be applied to shadow generation [14], [15], surface modeling [42], and visibility sorting in dynamic scenes [61].

Hybrid methods utilize information in both image space and object space. Zhang *et al.* [66] used both the bounding volume in object space and the occlusion map in image space to compute visibility. By combining hierarchical polygon tiling and a hierarchical visibility algorithm, Greene [28] developed a fast algorithm, based on Warnock's work [63], to render highly complex scenes. Naylor [43] proposed a 2D BSP image tree to represent the 2D projection of 3D polyhedral models in a continuous form. The representation proposed in his paper resembles ours, since we also construct a 2D BSP tree. While visibility in Naylor's approach is computed in 3D object space, we perform visibility sorting entirely in 2D image space using 2D input images rather than 3D geometric models.

2.2 Visibility in Image-based Computer Graphics

IBMR systems acquire images as inputs. In light field rendering [34] and the lumigraph method [27], the visibility problem is treated as an interpolation problem. When the image is equipped with depth, visibility issues can be resolved with depth-buffering [13]. If the images (sprites) are also associated with underlying 3D geometric models, visibility sorting [33], [54] can be performed in object space. Sometimes images are associated with coarse geometric primitives [18]. In this case, visibility can be computed with a standard depth-buffering algorithm.

However, there is no guarantee that the depth information or geometric models will always be available. Alternatively, disparity or depth can be recovered [31], [19] from stereo pairs or motion data. This information cannot be directly used in the geometry-based visibility algorithms above to resolve visibility. In some cases, an intermediate surface model (*pseudo-object*), has

to be reconstructed, introducing extra overhead. An alternate approach is to directly use the disparity to approximate the true visibility solution using a tailor-made rendering engine [35].

Starting from basic epipolar geometry, McMillan and Bishop [38] found that even when the true depth is not available, visibility-correct novel views can still be generated if the epipole is known. Figure 2 illustrates the basic idea. Positions $\mathbf{c}$ and $\mathbf{e}$ are the reference and the desired viewpoints respectively. Reference image I_c contains two pixels, i_1 and i_2, and their corresponding points in 3D space are $\mathbf{p_1}$ and $\mathbf{p_2}$ respectively. The projection of $\mathbf{e}$ onto the reference image is the epipole (positive in this example, denoted by a plus sign). When we generate the novel view at $\mathbf{e}$, even though exact positions of $\mathbf{p_1}$ and $\mathbf{p_2}$ may not be known, the correct visibility can still be ensured if we always draw i_1 before i_2. Notice that $\mathbf{p_1}$ cannot occlude $\mathbf{p_2}$ since i_1 is farther away from the positive epipole than i_2; thus, we can determine an occlusion-compatible drawing order for pixels from the position of epipole even though no absolute depth information is available.

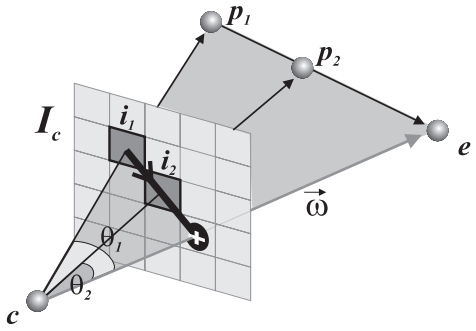


Fig. 2. The drawing order between two pixels that lie on the same epipolar line.

The same visibility algorithm is also used in later image-based rendering systems [53], [11], [36]. However, since this original occlusion-compatible drawing order is only applicable to pixel-sized image entities, the rendering is less efficient than it could be. Fu *et al.* [20] ex-

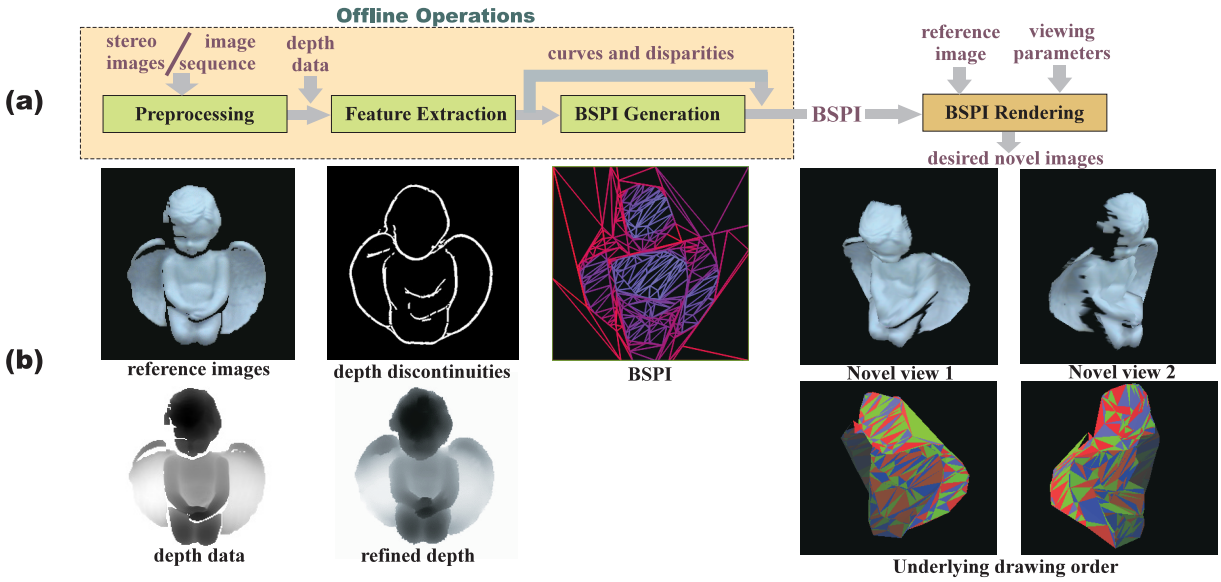


Fig. 3. Overall approach: (a) the flowchart and (b) a running example ANGEL. (Input image and initial depth map courtesy of the Signal Analysis and Machine Perception Laboratory, The Ohio State University).

tended this drawing-order strategy to arbitrarily-sized triangles and made use of texture-mapped triangles to speed up rendering. However, cycles may exist in the triangle drawing order in some rare cases, leading to incorrect visibility determination. The underlying reason is that the triangle-based ordering rule cannot guarantee a partial order.

3 OVERVIEW

Our system overview is shown in Fig. 3. The input consists of either a range image, a stereo pair, or a video sequence. A BSPI is then produced and rendered at novel views. The major steps of the system are as follows:

1. Automatic preprocessing estimates and cleans the given depth or disparity maps (Section 5.1)

2. Feature extraction by tensor voting (Section 5.2)

3. BSPI generation (Section 5.3)

4. BSPI rendering (Section 5.4)

The rest of this paper is organized as follows: In Section 4, we first explain how the BSPI ensures correct hidden surface removal, the main theoretical contribution of this paper. We detail the four steps in Section 5. Finally, we present results on a variety of input images in Section 6. A review of the tensor voting method is given in the Appendix.

4 The Binary-Space-Partitioned Image Representation

IBMR research motivates us to study the efficient rendering of image-based data that may not contain geometry information. To synthesize the desired images, we need to solve two sub-problems: (1) where image pixels move to, and (2) which pixel is in the front if more than one pixel moves to the same place.

Sub-problem (1) can be solved if depth information is available. The new pixel positions can be calculated, for example, by reprojection [13]. This problem can also be solved if the disparity of each pixel is known. One can warp the reference image pixel-by-pixel [38], [53], [11], [36] or triangle-by-triangle [20] while exploiting pixel coherence.

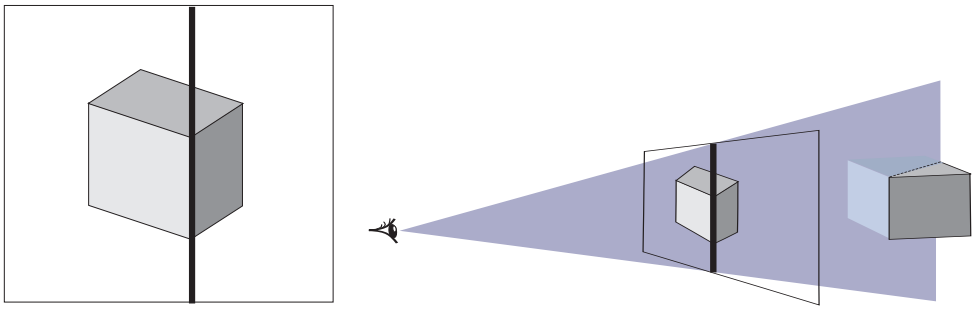
Sub-problem (2) also can be solved if accurate depth information is known. Depth-buffering is the simplest method. However, efficient rendering of high-resolution images depends on the performance of graphics hardware, as a per-pixel depth comparison is needed. If only disparity information can be recovered, the usual way is to fit a 3D surface (pseudo-object) and use this geometric surface for rendering. Recovering and rendering this intermediate pseudo-object introduces additional overhead.

In this paper, we focus on sub-problem (2), the visibility problem, with the assumption that sub-problem (1) can be handled by pixel reprojection or disparity interpolation. An efficient

algorithm based on the BSP visibility algorithm is proposed. No intermediate pseudo-object needs to be constructed and the algorithm is also independent of the resolution of the final desired image. It operates on convex polygonal primitives (not necessarily only triangles), hence taking advantage of the coherence between adjacent pixels. The efficiency of the BSP algorithm is evidenced by its common applications in computer game kernels [1], despite the wide availability of depth-buffer-based graphics hardware. We will show in the following sections that the proposed algorithm preserves the efficiency of the original geometry-based BSP algorithm and achieves real-time visibility sorting.

4.1 BSP in 2D

Every image reveals the visible surfaces seen from its corresponding viewpoint. Figure 4a shows the visible surfaces of a cube. Taking a 2D edge in the image plane, we can create an extended partition plane containing the reference viewpoint (the eye) and the edge (the bold line) on the image plane (Fig. 4b). There is a one-to-one correspondence between the partition plane and the 2D edge on the reference image. They are projection-equivalent with respect to the reference viewpoint. By recursively partitioning the 2D reference image into 2D convex



(a) (b)

Fig. 4. Forming the partition plane. (a) The reference image and (b) the partition plane formed by connecting the reference viewpoint and an edge in the image.

polygons (Fig. 5), we can create a corresponding BSP tree in 3D. We call the 2D binary-space-partitioned image the *BSPI*.

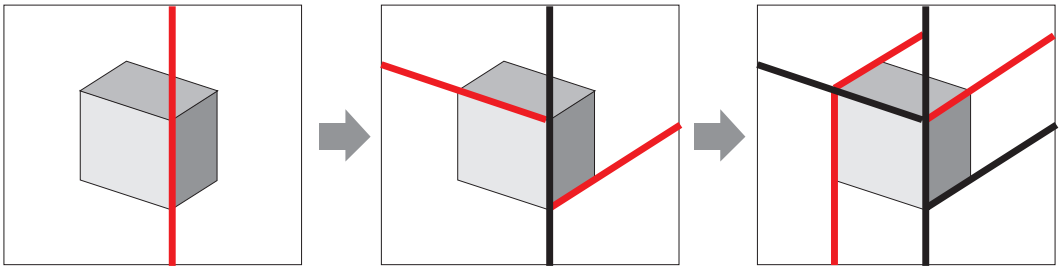


Fig. 5. Recursive partition of the 2D reference image.

Note that the original BSP visibility algorithm does not restrict the selection of partition planes. What we have done in the BSPI is to constrain all partition planes to intersect at the reference viewpoint as illustrated in Fig. 6, which depicts the top view of a simplified scene. The gray region indicates the space inside the field of view. The thick lines radiating from the reference viewpoint are the partition planes. The BSPI representation contains an implicitly embedded 3D BSP tree, and hence it inherits the corresponding visibility sorting capability.

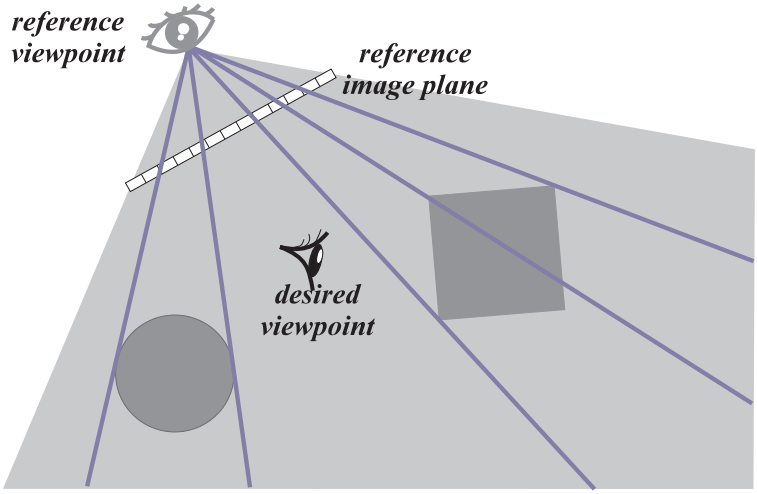


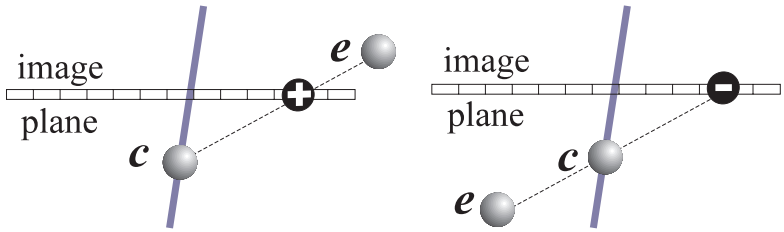
Fig. 6. Partition planes (thick dark (blue) lines) are all constrained to pass through the reference viewpoint (top view). Gray region indicates the space inside the field of view.

4.2 Visibility Sorting in 2D

Given the desired viewpoint as shown in Fig. 6, we need to warp the reference image to generate the related novel image. Since we have subdivided the reference image into a set of 2D convex polygons, warping is a process of moving and distorting these convex polygons. To ensure correct visibility, we can derive a drawing order from the BSPI. Just like traditional 3D BSP visibility, the visibility drawing order can be derived by traversing the BSP tree embedded in the BSPI. Recall that, during the tree traversal, we need to check to see on which side of the partition plane the desired viewpoint is found. Interestingly, we can carry out all such checks in 2D by projecting the desired viewpoint onto the reference image plane. The projection is the *epipole*.

To illustrate the idea, we show the 2D analogy (Fig. 7). In 2D, the 3D partition plane degenerates to a partition line (the thick blue line in Fig. 7). Similarly, the image plane degenerates to a line (symbolized by a horizontal line in Fig. 7). The desired viewpoint $\mathbf{e}$ is projected onto the reference image plane. In Fig. 7(a), the projection is a positive epipole, since the desired viewpoint is in front of the reference viewpoint $\mathbf{c}$ [1]. Since all partition planes (lines in 2D) are constrained to intersect at the reference viewpoint, we can always determine the drawing order by checking which side of the 2D partition lines the positive epipole resides on. Those convex polygons (projections of objects on the image plane) on the opposite side of the positive epipole should be drawn first as these objects can never occlude objects on the same side as the positive epipole.

[1]Whether the epipole is positive or negative depends on whether the projection is formed by intersection of positive epipolar ray (the vector from the reference viewpoint towards the desired viewpoint) or negative epipolar ray (the vector from the desired viewpoint towards reference viewpoint) with the reference image plane.



(a) (b)

Fig. 7. Positive and negative epipoles defined relative to the reference viewpoint (center of projection) and the desired viewpoint.

On the other hand, as shown in Fig. 7(b), when the desired viewpoint **e** is "behind" the reference viewpoint **c** (in the case of the negative epipole), the desired viewpoint **e** is actually on the other side of the partition plane (the thick blue line) because all partition planes must pass through the reference viewpoint **c**. Hence the drawing order is reversed. The rationale here is that by projecting everything (including the viewpoints and the partitioning lines/planes) onto the image plane, we can effectively compute the visibility order of image space triangles in the 2D image space.

4.3 Comparison with Related BSP Algorithms

Our BSPI method is inspired by McMillan's occlusion-compatible drawing order [39], which also makes use of epipolar geometry. In another classic work [4], drawing orders and grid splitting are used to ensure visibility correctness. Table I presents a comparison between the properties of the occlusion-compatible drawing algorithm and the BSPI.

TABLE I

COMPARISON BETWEEN THE OCCLUSION-COMPATIBLE DRAWING ALGORITHM AND THE BSPI

	McMillan's method [39]	BSPI
Drawing order	Based on epipolar geometry	Based on BSP in image space
Drawing primitive	Pixel	Polygon
Number of primitives	Fixed (image size)	*O*(image size), depending on how much image space coherence is exploited by tensor voting. The number of polygons is usually much smaller than the number of pixels in the image.
Depth buffering	No	
Hardware acceleration	Hardware-accelerated rendering of points	Hardware-accelerated rendering of textured polygons
Gaps in rendered views	Yes (due to forward mapping)	No
Multiple views	Possible by LDI or LDI trees	Layered BSPI

5 THE BSPI RENDERING SYSTEM

Now that we have outlined the theory behind the BSPI method, we next describe the four stages of the BSPI system implementation. The system flowchart in Fig. 3 provides an overview, and we use the ANGEL image in that figure as the running example to facilitate discussion.

5.1 Preprocessing (depth estimation and noise filtering)

If depth information is unavailable, we perform a preprocessing step to estimate the scene depth. Note that depth is used in preprocessing only, and no depth sorting is needed in subsequent novel-view rendering. We consider two common cases for which we have images but no

initial depth information: 1) a stereo pair, and 2) an image sequence. Computer vision algorithms for stereo and motion-sequence data can be applied to estimate the corresponding depth maps. Alternatively, if a noisy initial depth map is available (see, e.g., ANGEL's noisy depth data, Fig. 3b), it can be cleaned up by rejecting outliers and inferring missing details (see, e.g., ANGEL's refined depth map, Fig. 3b).

In this work, *tensor voting* [40] is used to estimate the depth or disparity maps from stereo or motion data, and to clean up noisy depth maps. We will not repeat here the details of customizing general voting frameworks to infer depth maps from stereo and motion data, but refer interested readers to [32] for depth from stereo, and [25] for depth from motion. We will, however, supply an intuitive illustration of the generic methodology that is used to infer missing details and reject outliers. Additional information on tensor voting can be found in [40]. Detailed pseudocode for the methods we employ is provided in the appendix; the core C++ implementations of the pseudocode segments (about 200 lines of code) are available in the library TVlib (see Section 7). The tensor voting algorithm can be implemented efficiently, and it runs in $O(nk)$ time, where n is the number of points (pixels), and k is the size of neighborhood, or the scale of the analysis.

Tensor voting is a computational framework for feature extraction and segmentation. Tensors are used for data representation, and they "communicate" with each other via a voting algorithm. In essence, we want to answer the following question: given a point P with a finite neighborhood, what is the most likely normal direction at P, if there is any? Consider an analogy in particle physics, two particles vibrate and each particle emits a waveform to communicate with the other particle. Their vibrations reinforce each other if the particles are vibrating at *compatible* frequencies, resulting in a maximum (or resonance). The *ball voting field*, described in the appendix, is used to mimic this communication process.

Figure 8a shows a scenario with an outlier E and a missing data point. Suppose points A, B, C, D, F, G, H, and I all lie on an underlying smooth curve. Inspired by the particle physics analogy, we want the tangent directions at each point to be *compatible* to produce a smooth curve; for example, the tangent directions at all points on a straight line should be equal.

How do we find the compatible tangent direction, say θ, at a particular point C? In tensor voting, all points communicate with each other by the ball voting field, which suggests desirable tangent directions. After C has collected a set of suggested tangent directions, principal component analysis is performed to determine an optimal θ. The collected votes can be understood as a

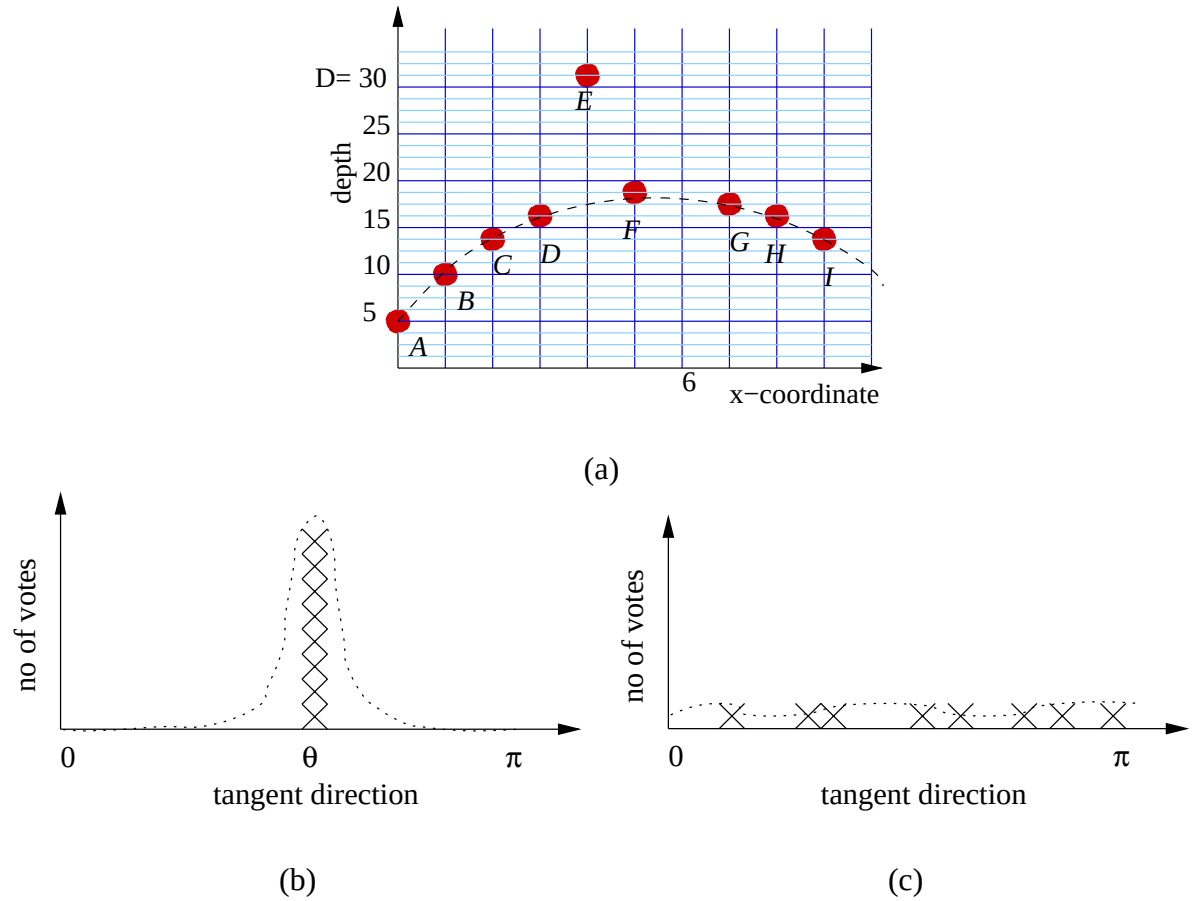


Fig. 8. (a) Rejecting outliers and inferring missing details by tensor voting. (b) Histogram for point C after tensor voting: θ is the *compatible* tangent direction at point C producing a smooth curve. (c) Histogram for point E after tensor voting: no maximum is produced since E is an outlier.

histogram: in Fig. 8b, a maximum occurs at θ after vote collection for the point C (the situation is of course oversimplified in this illustration; in practice, we see a distribution of angles such as that suggested by the dotted curve).

To detect outliers, we look for atypical tensor voting histograms like that shown in Fig. 8c. Since E receives a set of incompatible directions from the other eight points, no salient maximum can be found in the histogram.

At $x = 6$, there is a missing data point in Fig. 8a. All the eight points cast directional votes to the set of locations $\{(x, d)|d \in [0, D]$ is the set of quantized depths$\}$. By the continuity constraint imposed by the ball voting field, $(6, 17)$ is the solution. A salient maximum should be found in its corresponding histogram after tensor voting. Consequently, we not only reject outliers, but can also fill in missing details, and infer probable tangent (or normal) directions.

5.2 Feature Extraction

Once a (cleaned) depth map is available, we perform feature extraction on the depth map, so that illumination, colors, and textures are no longer problematic. The output feature curves indicate the loci of the depth discontinuities (see ANGEL's depth discontinuities, Fig. 3), along which we may choose to place the partition lines of the BSPI.

Next, we illustrate the use of tensor voting to localize feature curves (for details see [60]). We use the RENAULT example in Fig. 9 to illustrate an underlying depth surface that is piecewise smooth but not closed. Our goal is to label the *depth discontinuity curves* automatically (drawn in bright red in Fig. 9).

Consider a finite neighborhood around a point $P = (x, y, d)$, where (x, y) are the image coordinates and d is the depth. Note that the normal direction at P, and thus its tangent plane, is

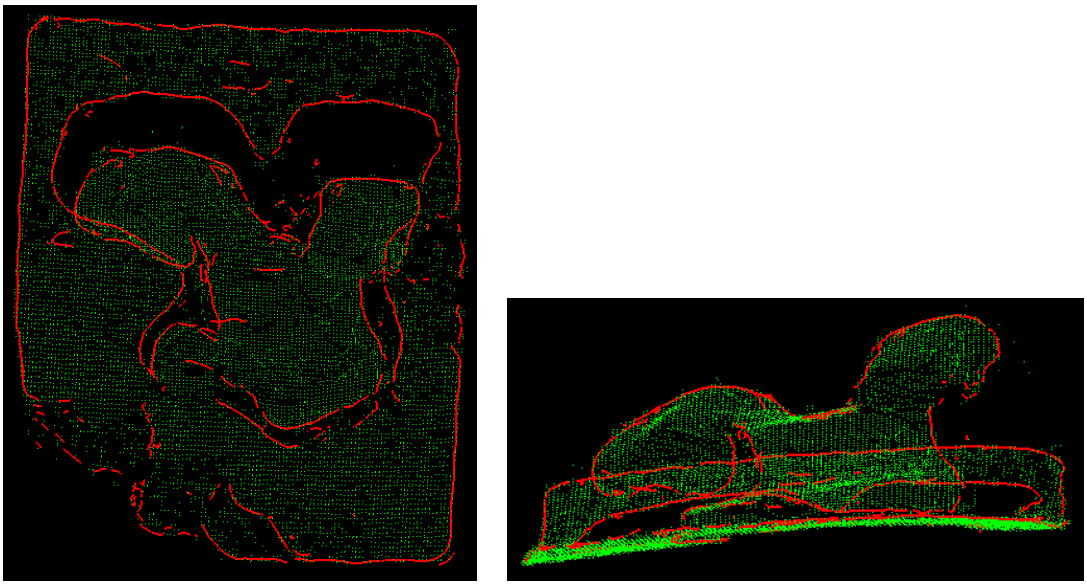


Fig. 9. Two views of the depth surface of the RENAULT model. The bright red curves are depth discontinuity and boundary curves.

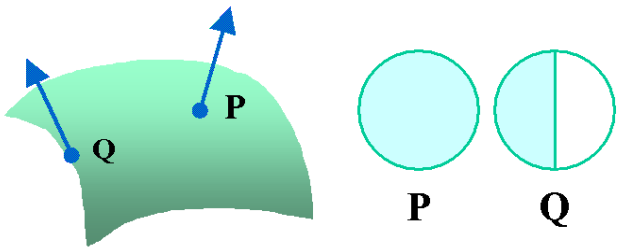


Fig. 10. Automatic depth discontinuity detection by tensor voting: uneven distribution of surfel votes.

available after preprocessing. We collect the points within P's neighborhood such that they lie on a smooth surface. This collection is projected onto P's tangent plane. To check if a neighboring point is smoothly connected to P, we arrange for the points to communicate with one another using the *stick voting field* (defined in the appendix).

Finally, we consider the vote distribution of the projected points. If the distribution is uneven, we label the point as a boundary point (Fig. 10).

5.3 BSPI Generation

Since we apply tensor voting in the 3D *x-y-d* volume in order to enforce surface smoothness, we perform the following steps before building a BSPI in the 2D image domain (see, e.g., ANGEL's BSPI, Fig. 3):

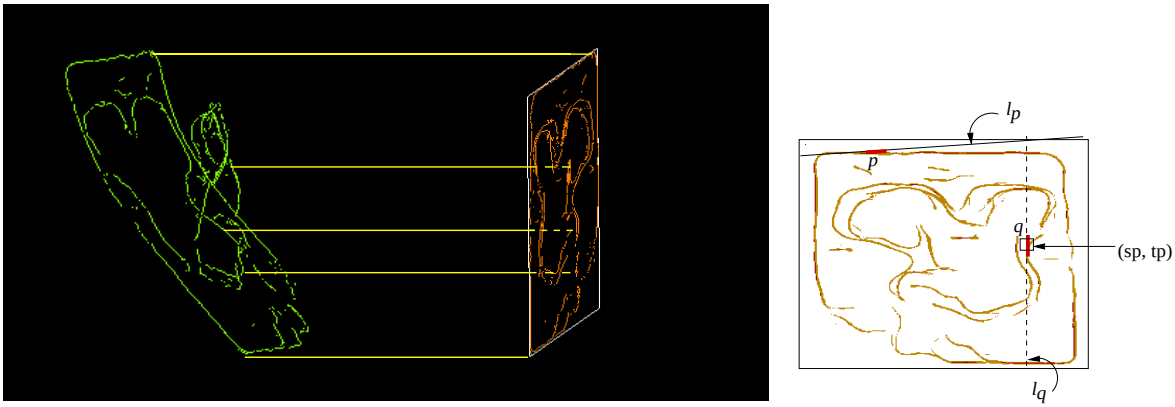
1. Project and quantize the discontinuity curves obtained in Section 5.2 onto the 2D image domain. Let the projected set be $X = \{(x, y)\}$ as illustrated in Fig. 11a.

2. Apply tensor voting to X, and collect directions at *every* pixel in the image domain. This process generates our dense 2D *CMap* map. Each pixel in a *CMap* has an associated tuple $(s_p, \hat{t}_p)$, where s_p is a scalar indicating the saliency or confidence level for the presence of a salient curve, and $\hat{t}_p$ is a unit vector indicating the 2D tangent direction if a curve exists (see Fig. 11b).

Before calling the recursive procedure **BSPI** whose pseudocode is given below, we initialize the structure *bsptree* to be empty, and set the structure *polygon* to be the rectangular 2D image domain.

BSPI (*polygon*,*bsptree*,*CMap*) {
  Pick a candidate set Y:
    $Y = \{p = (s_p, \hat{t}_p) | s_p \geq$ some saliency value and $p \in$ *polygon*$\}$
  if ($Y = \emptyset$) return
  Compute $score_p$ for all $p \in Y$ (*score* is defined below)
  Partition *polygon* into $P_1 \cup P_2 \cup \{\ell\}$ at pixel $q \in Y$,
    such that $score_q$ is maximum.
    ℓ is the partition line, whose gradient is given by $\hat{t}_q$.
  bsptree $\leftarrow$ *bsptree* $\cup \{\ell\}$
  Inhibit pixels in *CMap* close to ℓ
  BSPI (P_1,*bsptree*,*CMap*)
  BSPI (P_2,*bsptree*,*CMap*)
}

Therefore, Y is the set of pixels with high curve saliencies s_p. In other words, partition lines should pass through these pixels in order to respect the depth discontinuities. In each recursive



(a) (b)

Fig. 11. (a) Projecting the discontinuity curves. (b) *CMap* is produced after tensor voting. Here, $score_p > score_q$ since $\ell_q \cap R \neq \emptyset$. Here ℓ_p is chosen as the partition line.

step, each $p \in Y$ generates a candidate partition line ℓ_p. We quantize ℓ_p in the image domain and set $R = I - Y$, where I is the set of all image pixels. R is therefore the set of region pixels, and corresponds to smooth depth surfaces. We penalize ℓ_p if it trespasses the interior of an object, i.e., if $\ell_p \cap R \neq \emptyset$. Therefore, $score_p$ is defined as

$$score_p = \alpha \sum_{e \in E} d_{p,e} e - \beta \sum_{r \in R} r \ ,$$

where $\alpha, \beta > 0$. $d_{p,e}$ is the inner product of ℓ_p and $\hat{t}_e$, indicating their orientation consistency.

When we have computed ℓ that gives the maximum score, an envelop of pixels surrounding the quantized ℓ is inhibited in order to avoid producing elongated partitions in later recursions. Convex polygons at leaf nodes are then triangulated to support the rendering operation.

5.4 BSPI Rendering

Rendering a BSPI consists of two steps: visibility sorting and warping. Visibility sorting begins at the root node of the BSPI and the order of subtree traversal depends on the location

of the novel view, as discussed in Section 4. Note that since the BSPI is a static data structure, no write-back to memory is needed when resolving visibility as is the case for depth buffering. In contrast, straightforward comparison at internal nodes is sufficient to produce the correct warping order. Two novel views of the rendered result and the underlying drawing orders of triangles of ANGEL are shown in Fig. 3, using a color code to be explained below.

During the BSP tree traversal, whenever we visit a leaf node, we warp and draw the polygon that corresponds to the current leaf node. Warping can be done either using the 2D warping equation [36], or by reprojecting vertices onto the image planes of the novel view (Fig. 12). Consequently, we can draw the warped polygons with the reference images texture-mapped onto them. Finally, we apply a standard feathering technique that adjusts the alpha value at pixels along edge discontinuities to avoid undesirable artifacts and enhance the image quality.

6 RESULTS

To demonstrate the effectiveness and efficiency of the proposed method, we create BSPI's from three different kinds of real images: single-view range images with corresponding color images (Fig. 16), stereo image pairs without depth (Fig. 17), and motion sequences without

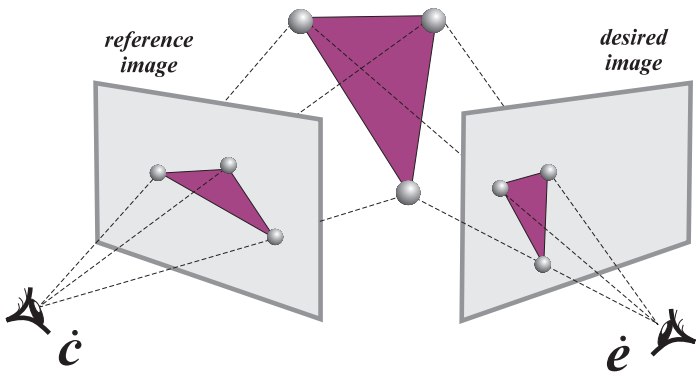


Fig. 12. Reprojecting vertices of polygons to synthesize novel views.

depth (Fig. 18). Readers are encouraged to visit the webpages listed in section 7 for the video that explains our algorithm with running examples and animations.

Figure 16a shows the single-view image of a human FACE with its given depth in Fig. 16b (captured by a laser range finder). Although the captured depth data is quite accurate, holes exist due to the scanner's inability to capture surfaces with particular reflectivities. Tensor voting is applied to fill these gaps and remove noise. The resulting BSPI is shown in Fig. 16c. To illustrate the order of partition steps, we color-code the partition lines. Partition lines generated in earlier iterations are coded in red (brighter), while those in later iterations are in blue (darker). As shown in Fig. 16c, the first few partitions are done at the silhouette of the face, where higher depth discontinuities are located. Figures 16d–e show the rendered novel views. Their corresponding drawing orders are illustrated in Fig. 16f–g by color-coding triangles. Triangles that are to be drawn earlier are darker, while those drawn at a later stage are brighter. Hence, foreground triangles are brighter than those in the background. Note how the nose occludes the other parts of the face in the rendered images. The average frame rate for the rendering shown under the figures is measured on a PC with a Pentium III 750MHz processor and 128MB RAM without any graphics accelerator installed. Thus our method offers real-time rendering on ordinary PCs. It is worth noting that due to the hierarchical nature of the BSPI, we can naturally introduce level-of-detail control to further speed up the rendering. Another example of a range image is the angel sculpture ANGEL shown in Fig. 3b.

Figure 17 shows an input stereo image pair for the RENAULT piece, a manufacturing part. No depth is given. Disparity is recovered from the stereo pair. The recovered depth may not be Euclidean, but can be used for generating novel views from different viewpoints. As before, we color-code the partition lines (Fig. 17c) and the triangles (Fig. 17f–g) to show the partition

process and drawing orders, respectively.

Fig. 18 demonstrates the use of multiple BSPIs to handle occlusion inherent in motion data, the FLOWER GARDEN sequence in our case. A portion of the flower garden scene is occluded by a tree trunk in the reference frames. The initial depth map is obtained from three frames [25]. First, a layered and textured description involving the foreground and the background is obtained by tensor voting. Texture synthesis in the presence of occlusion is detailed in [30]. Note that this texture synthesis preprocessing step is also automatic. Two BSPIs, which we call *layered BSPIs*, are obtained. During rendering, the background BSPI is traversed first, followed by the foreground BSPI. Our system can be readily scaled up to handle additional layered BSPIs. Since the drawing order has been resolved by the layered descriptions, rendering multiple BSPIs does not need any depth sorting. Note that our method alleviates the problem of "disocclusion," when a previously occluded part of the scene becomes visible at a novel viewpoint. No gaping hole is observed in our result. The same example has been used by many researchers (layered representation in [62] and sprites in [5], among others). To synthesize a novel view, the depth layers or sprites are rendered at a given viewpoint. No depth sorting is required in the BSPI rendering approach.

Note that using tensor voting, constructing a BSPI from the raw depth or disparity data takes only a few minutes. This varies according to the noisiness and complexity of the dataset. Like the BSPI warping, all the processing is done in 2D, with the exception of the feature curve inference. The latter is performed in disparity space in order to enforce the local smoothness constraint. Still, this can be done very efficiently since we do not fit any pseudo-object or disparity surface for curve inference. The preprocessing step is performed only once, including the BSPI generation. Once the BSPI is generated, it can be used by the BSPI warper to yield

	ANGEL	FACE	FLOWER GARDEN (background)	FLOWER GARDEN (foreground)
No. of pts in feature extraction (Section 5.2)	14,670	20,281	76,161	10,922
Time for feature extraction	72 sec	95 sec	251 sec	39 sec
Time for BSPI generation (Section 5.3)	12 sec	26 sec	103 sec	5 sec
Total time for feature extraction and BSPI generation	84 sec	121 sec	354 sec	44 sec
Height of the BSP tree generated	33	44	38	24

TABLE II

SAMPLE PREPROCESSING TIMES FOR BSPI GENERATION.

novel views efficiently.

Table II tabulates some sample preprocessing times of the results we presented. All preprocessing times were measured on an IBM ThinkPad PIII 850 MHz with 512M RAM, without any hardware acceleration. The mean preprocessing rate for depth estimation and noise filtering (Section 5.1) is 10,000 points per minute. Processing times for feature extraction (Section 5.2) and BSPI generation (Section 5.3) are shown. These typical times show that our automatic preprocessing, which is run once, takes only a few minutes. The height of the BSP tree generated is also reported in the table.

7 WEB AVAILABILITY

Interested readers are encouraged to visit the following web-sites for an accompanying video explaining the BSPI algorithm with running examples presented in this paper:

`http://www.cs.indiana.edu/~cwfu/papers/bspi/bspi.html (site 1)`

`http://www.cse.cuhk.edu.hk/~ttwong/papers/bspi/bspi.html (site 2)`

For the detailed implementation of tensor voting, readers can obtain the core library of C++ code from `http://www.cs.ust.hk/~cktang/TVlib.zip`.

8 CONCLUSION

In this paper, we present the BSPI – a novel IBMR approach to represent and render range images, stereo image pairs, and motion data. No intermediate pseudo-objects need to be constructed for the rendering. The BSPI representation implicitly embeds a projection-equivalent 3D BSP tree into the 2D image plane, and thus actually supports a version of the general 3D BSP visibility algorithm. We utilize epipolar geometry to perform visibility sorting completely in 2D image space rather than 3D object space. Our results demonstrate the efficiency and robustness of the method. The BSP representation of images also facilitates efficient image-based data compression [49] and image retrieval [47]. To handle noise in the input data, we extend the tensor voting algorithm for the data cleanup process, and incorporate the results into the recursive BSP partitioning process. This technique enhances the noise tolerance in the BSPI partitioning stage. To handle disocclusion, we combine multiple BSPI's from the same reference viewpoint, giving a layered representation of the scene. We also obtain this representation using tensor voting in the automatic preprocessing step.

In the future, we plan to extend the layering method to combine multiple BSPI's from multiple viewpoints. Layering techniques similar to the Layered Depth Image (LDI) method [53] can be exploited to combine multiple BSPI's corresponding to different viewpoints. This is useful for fast object surface reconstruction using data from single-view laser range scanners. Currently we assume surfaces are Lambertian; view-dependent texture mapping could be used to remove this limitation. In addition, it would be beneficial to apply the BSPI partitioning method directly to the optic-flow field, thus avoiding the loss of information when converting to depth.

APPENDIX

A TENSOR REPRESENTATION

A tensor encapsulates both direction certainty *and* uncertainty. Consider a point in 2D space. It belongs to a curve (absolute certainty in a single direction), is a point junction (absolute *un*certainty in all directions), or is an outlier. The 2D stick and ball tensors are used respectively to encode the two extreme cases. The whole continuum can therefore be described as an ellipse, or equivalently, as a second order symmetric tensor in 2D: mathematically, it is a 2×2 eigensystem, with two unit eigenvectors $\hat{e}_1$ and $\hat{e}_2$, and two eigenvalues $\lambda_1 \geq \lambda_2$ (see Fig. 13).

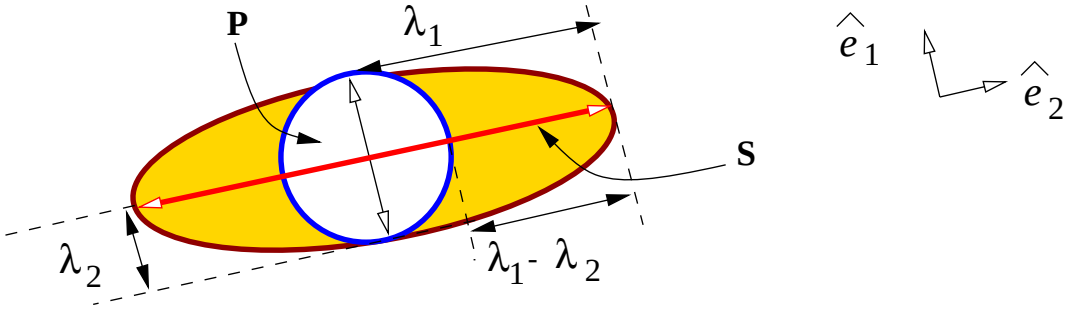


Fig. 13. Ellipse: a second order symmetric tensor in 2D.

The eigenvectors represent the tangent direction, and the eigenvalues indicate the magnitudes of orientation (un)certainties, since the latter encode the size of the ellipse. The equation of a second order symmetric tensor is $(\lambda_1 - \lambda_2)\mathbf{S} + \lambda_2\mathbf{B}$ where $\mathbf{S} = \hat{e}_1\hat{e}_1^T$ defines a stick tensor, and $\mathbf{B} = \hat{e}_1\hat{e}_1^T + \hat{e}_2\hat{e}_2^T$ defines a ball tensor. If normal or tangent information is unavailable initially, we encode the input points as ball tensors. Take the 2D case as example. Then, each point $(x, y)^T$ is associated with an eigensystem with $\lambda_1 = \lambda_2 = 1$, $\hat{e}_1 = [1\ 0]^T$, $\hat{e}_2 = [0\ 1]^T$. After encoding, the input points communicate with one another via the ball voting field.

B THE STICK AND BALL VOTING FIELDS

We want to answer the following question: *for a given point P in space, and normal $\vec{N}$ at origin, what is the most likely normal at P if there is a smooth surface interpolating O (the origin) to P, and normal to $\vec{N}$?* Fig. 14 illustrates the situation in 3D. We claim that the osculating

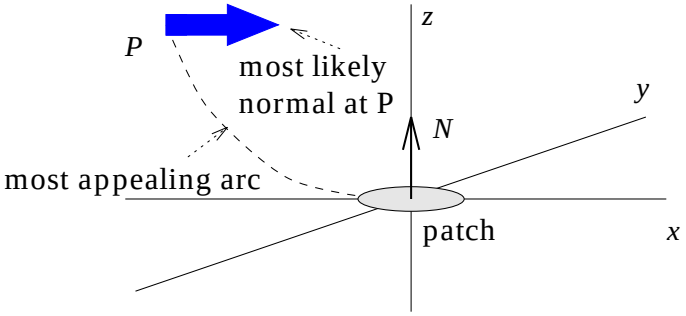


Fig. 14. The design rationale of the stick voting field.

circle connecting O and P is the most likely curve, since it keeps the curvature constant along the hypothesized circular arc. See [40] for detailed mathematical derivation. The most likely normal direction at P is thus given by the normal to the osculating circle at P. The length of the normal vector at P, which represents the surface saliency of the vote, is inversely proportional to the arc length OP, and also to the curvature of the underlying circular arc. In spherical coordinates, the decay of the field takes the following form:

$$\overline{DF}(r, \varphi, \sigma) = e^{-(\frac{r^2+c\varphi^2}{\sigma^2})} \tag{1}$$

where r is the arc length OP, φ is the curvature, and σ is the size of neighborhood, or scale of analysis. The set of normals at all P's in the 2D space constitutes the stick voting field V_S.

The ball voting field V_B is obtained by rotating and integrating vote contributions of the 2D *stick voting field*, since all directions are equally likely: $V_B = \int_0^{\pi} V_S d\theta$. Fig. 15 shows the shapes, directions, and strengths of stick and the ball voting fields in 2D.

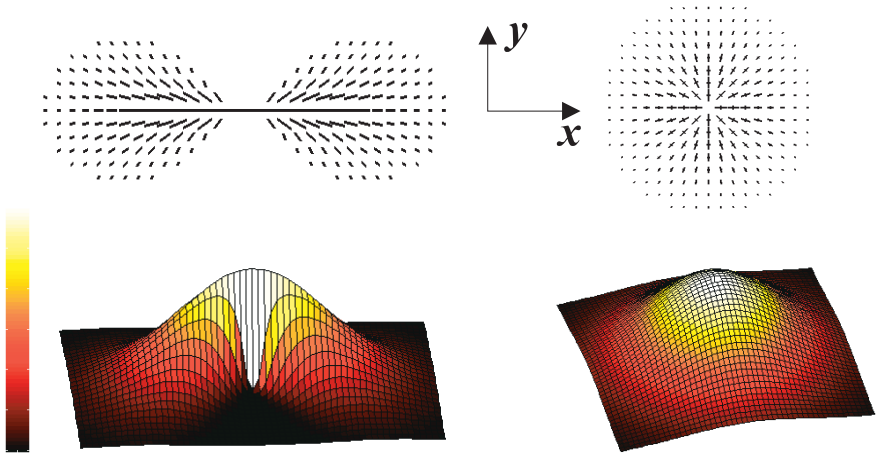


Fig. 15. The directions and strengths of the stick and ball voting fields.

C ALGORITHMS OF 2D TENSOR VOTING

This section describes the general tensor voting algorithm [57]. The voter makes use of GENTENSORVOTE to cast a tensor vote to vote receiver (votee). Normal direction votes generated by GENNORMALVOTE are accumulated using COMBINE. A 2×2 *outTensor* is the output. The votee thus receives a set of *outTensor* from voters within its neighborhood. All *outTensor*'s are summed up by tensor addition, which is simply ordinary 2×2 matrix addition. This resulting matrix is equivalent to an ellipse.

ACKNOWLEDGMENTS

We would like to thank Dr. Michael S. Brown for his audio help in making the video, which is available at our web-sites. Also, we would like to thank Gaucher, Lee, and Medioni as well as the Signal Analysis and Machine Perception Laboratory, the Ohio State University for various datasets we used in this research work. This research was supported in part by the Hong Kong Research Grant Council Co-operative Research Centres (CRC) Scheme (Project no. CRC 4/98) and RGC Earmarked Grants (Project No. CUHK 4186/00E, HKUST6089/99E, and

Algorithm 1 GENTENSORVOTE (voter,votee)

```
/* It uses GENNORMALVOTE to compute the most likely normal
direction vote at the votee. Then, plate and ball tensors are
computed, by integrating the resulting normal votes cast by voter. */
for all 0 ≤ i, j < 2, outTensor[i][j] ← 0
voterSaliency[0] ← voter[λ1] − voter[λ2]
voterSaliency[1] ← voter[λ2]
/* Generate the stick component */
if (voterSaliency[0] > 0) then
   vecVote ← GENNORMALVOTE (voter,votee)
   /* Compute stick component */
   COMBINE (outTensor,vecVote)
end if
/* Compute ball component */
if (voterSaliency[1] > 0) then
   /* count is a sufficient number of samples uniformly distributed on a unit circle */
   while (count ≠ 0) do
      /* Generate a random point on a unit circle. */
      voter[direction] ← random[direction] ← GENRANDOMUNIFORMPT()
      vecVote ← GENNORMALVOTE (voter,votee)
      COMBINE (outTensor, vecVote, voterSaliency[1])
      count ← count − 1
   end while
end if
return outTensor
```

HKUST6193/02E).

REFERENCES

[1] M. Abrash. Inside quake: Visible-surface determination. *Dr. Dobb's Sourcebook*, pages 41–45, January/February 1996.

[2] Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 107–116, Jan. 1998.

[3] J. M. Airey, J. H. Rohlf, and F. P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In *ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics*, pages 41–50, 1990.

[4] D. P. Anderson. Hidden line elimination in projected grid surfaces. *ACM Transactions on Graphics*, 1(4):274–288, October 1982.

[5] S. Baker, R. Szeliski, and P. Anadan. A layered approach to stereo reconstruction. In *Proceedings of IEEE Computer Vision and Pattern Recognition 1998 (CVPR'98) Conference*, 1998.

Algorithm 2 GENNORMALVOTE (voter, votee)
A vote (vector) on the most likely normal direction is returned.

$v \leftarrow$ votee[position] $-$ voter[position]
$/*$ voter and votee are connected by high curvature? $*/$
if (angle(voter[direction],v) $< \pi/4$) **then**
  return NULL {smoothness constraint violated}
end if
stickvote[position] $\leftarrow$ votee[position]
$/*$ voter and votee on a straight line, or voter and votee are the same point $*/$
if (angle(voter[direction],v) $= \pi/2$) **or** (voter = votee) **then**
  stickvote[direction] $\leftarrow$ voter[direction]
  stickvote[length] $\leftarrow e^{-\frac{r^2}{\sigma^2}}$ {equation (1)}
  return stickvote
end if
Compute center and radius of the osculating hemisphere
$/*$ assign stick vote $*/$
stickvote[direction] $\leftarrow$ center $-$ votee[position]
stickvote[length] $\leftarrow e^{-}(\frac{r^2+c\kappa^2}{\sigma^2})$ {equation (1)}
return stickvote

Algorithm 3 COMBINE (tensorvote, stickvote, weight)

$/*$ It performs tensor addition, given a stick vote. $*/$
for all i, j such that $0 \leq i, j < 2$ **do**
  tensorvote[i][j] $\leftarrow$ tensorvote[i][j] + weight $\times$ stickvote[i] $\times$ stickvote[j]
end for

[6] De Berg, De Groot, and Overmars. Perfect binary space partitions. *CGTA: Computational Geometry: Theory and Applications*, 7:81–91, Jan. 1997.

[7] W. Bouknight. A procedure for generation of three-dimensional halftoned computer graphics presentations, 1970.

[8] W. Bouknight and K. Kelly. An algorithm for producing halftone computer graphics presentations with shadows and movable light sources, 1970.

[9] T. Cassen, K. R. Subramanian, and Z. Michalewicz. Near-optimal construction of partitioning trees by evolutionary techniques. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface '95*, pages 263–271. Canadian Human-Computer Communications Society, 1995.

[10] Ed. E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, 1974.

[11] C.-F. Chang, G. Bishop, and A. Lastra. LDI tree: A hierarchical representation for image-based rendering. *SIGGRAPH '99 Conference Proceedings*, pages 291–298, August 1999.

[12] H.-M. Chen and W.-T. Wang. The feudal priority algorithm on hidden-surface removal. *Proceedings of SIGGRAPH 96*,

pages 55–64, August 1996.

[13] S. Eric Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[14] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. In *SIGGRAPH '89 Conference Proceedings*, volume 23, pages 99–106, July 1989.

[15] N. Chin and S. Feiner. Fast object-precision shadow generation for areal light sources using BSP trees. In *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, pages 21–30, March 1992.

[16] Joao Luiz Dihl Comba. *Kinetic Vertical Decomposition Trees*. PhD thesis, Dept. of CS, Stanford U., 1999.

[17] G. Crocker. Invisibility coherence for faster scan-line hidden surface algorithms. In *SIGGRAPH '84 Conference Proceedings*, volume 18, pages 95–102, July 1984.

[18] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH '96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[19] O. Faugeras and L. Robert. What can two images tell us about a third one? Technical report, INRIA, July 1993.

[20] C.-W. Fu, T.-T. Wong, and P.-A. Heng. Computing visibility for triangulated panorama. In *in Proceedings of the 10th Eurographics Rendering Workshop*, Granada, Spain, June 1999. Eurographics.

[21] H. Fuchs, G. Abram, and E. Grant. Near real-time shade display of rigid objects. In *SIGGRAPH '83 Conference Proceedings*, volume 17, pages 65–72, July 1983.

[22] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80 Conference Proceedings*, volume 14, pages 124–133, July 1980.

[23] T. A. Funkhouser, C. H. Séquin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. In *ACM SIGGRAPH Special Issue on 1992 Symposium on Interactive 3D Graphics*, pages 11–20, 1992.

[24] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In James T. Kajiya, editor, *SIGGRAPH '93 Conference Proceedings*, volume 27, pages 247–254, August 1993.

[25] L. Gaucher and G. Medioni. Accurate motion flow estimation with discontinuities. In *Seventh International Conference on Computer Vision (ICCV'99)*, pages 695–702, September, 1999.

[26] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[27] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'96*, pages 43–54, August 1996.

[28] N. Greene. Hierarchical polygon tiling with coverage masks. In *SIGGRAPH '96 Conference Proceedings*, pages 65–74, August 1996.

[29] N. Greene and M. Kass. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

[30] J. Jia and C.-K. Tang. Texture synthesis in the presence of discontinuities from noisy data. In *Eurographics Rendering Workshop 2002 (submitted)*, 2002.

[31] S. Laveau and O. Faugeras. 3-D scene representation as a collection of images. In *Proceedings of the Twelfth International Conference on Pattern Recognition (ICPR '94)*, pages 689–691, Jerusalem, Israel, October 1994.

[32] M.-S. Lee and G. Medioni. Inferring segmented surface description from stereo data. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition 1998 (CVPR'98), Santa Barbara, California*, pages 346–52, June, 1998.

[33] J. Lengyel and J. Snyder. Rendering with coherent layers. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'97*, pages 233–242, August 1997.

[34] M. Levoy and P. Hanrahan. Light field rendering. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'96*, pages 31–42, August 1996.

[35] M. Lhuillier and L. Quan. Image interpolation by joint view triangulation. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 1999.

[36] W. R. Mark and G. Bishop. Memory access patterns of occlusion-compatible 3d image warping. In *Proceedings of the 1997 Siggraph/Eurographics Workshop on Graphics Hardware*, pages 35–44, August 1997.

[37] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Transactions on Graphics*, 6(1):19–28, January 1987.

[38] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'95*, pages 39–46, August 1995.

[39] Leonard McMillan. *An Image-based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, 1997.

[40] G. Medioni, M.-S. Lee, and C.-K. Tang. *A Computational Framework for Feature Extraction and Segmentation*. Elseviers Science, Amsderstam, 2000.

[41] K. Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics (Proceedings of SIGGRAPH 89)*, 23(3):379–388, July 1989. Held in Boston, Massachusetts.

[42] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. In *SIGGRAPH '90 Conference Proceedings*, volume 24, pages 115–124, August 1990.

[43] B. F. Naylor. Partitioning tree image representation and generation from 3d geometric models. *Graphics Interface '92*, pages 201–212, May 1992.

[44] B. F. Naylor. Constructing good partitioning trees. In *Graphics Interface '93*, pages 181–191, May 1993.

[45] M. E. Newell, R. G. Newell, and T. L. Sancha. A solution to the hidden surface problem. In *Proceedings of the ACM Annual Conference*, volume I, pages 443–450, Boston, Massachusetts, 1972.

[46] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete & Computational Geometry*, 5:485–503, 1990.

[47] G. Qiu and S. Sudirman. Color image coding, indexing and retrieval using binary space partitioning tree. In *IEEE International Conference on Image Processing*, October 2001.

[48] H. R. Rabiee, R. L. Kashyap, and H. Radha. Multiresolution image compression with bsp trees and multi-level block truncation coding. In *IEEE 2nd International Conf. on Image Processing, Washington D.C.*, pages 600–603, 1995.

[49] H. Radha, M. Vetterli, and R. Leonardi. Image compression using binary space partitioning trees. *IEEE Transactions on Image Processing*, 5(12):1610–1624, December 1996.

[50] H. Rahda, R. Leonardi, M. Vetterli, and B. F. Naylor. Binary space partitioning tree representation of images. *Visual Communications and Image Representation*, 2(3):201–221, Sept. 1991.

[51] G. Schaufler and M. Priglinger. Efficient displacement mapping by image warping. In *the 10th Eurographics Workshop on Rendering*, pages 175–186, 1999.

[52] S. Sechrest and D. P. Greenberg. A visible polygon reconstruction algorithm. *ACM Transactions on Graphics*, 1(1):25–42, January 1982.

[53] J. Shade, S. Gortler, L.-W. He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, July 1998.

[54] J. Snyder and J. Lengyel. Visibility sorting and compositing without splitting for image layer decomposition. *Proceedings of SIGGRAPH 98*, pages 219–230, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[55] K. R. Subramanian and B. F. Naylor. Converting discrete images to partitioning trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):273–288, 1997.

[56] O. Sudarsky and C. Gotsman. Output-sensitive visibility algorithms for dynamic scenes with applications to virtual reality. *Computer Graphics Forum*, 15(3):249–258, August 1996.

[57] C.-K. Tang, G. Medioni, and M.-S. Lee. N-dimensional tensor voting, and application to epipolar geometry estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-23(8):829–844, August 2001.

[58] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. *Proceedings of SIGGRAPH 93*, pages 239–246, August 1993.

[59] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 61–69, July 1991. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.

[60] W.-S. Tong and C.-K. Tang. First order tensor voting, and application to 3d scale analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition 2001 (CVPR'01), Kauai, Hawaii*, December, 2001.

[61] E. Torres. Optimization of the binary space partition algorithm (bsp) for the visualization of dynamic scenes. *Eurographics '90*, pages 507–518, September 1990.

[62] J.W.A. Wang and E. H. Adelson. Layered representation for motion analysis. In *Proceedings of IEEE Computer Vision and Pattern Recognition 1993 (CVPR'93) Conference*, pages 361–366, 1993.

[63] J. Warnock. A hidden-surface algorithm for computer generated half-tone pictures. Technical Report TR 4-15, NTIS AD-753 671, University of Utah, Computer Science Department, 1969.

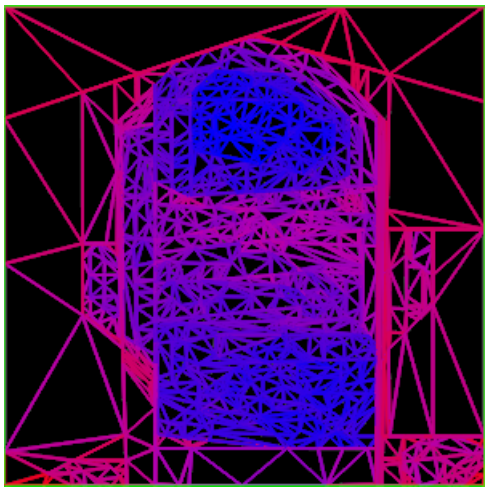
[64] G. S. Watkins. A real-time visible surface algorithm. Technical Report UTECH-CSc-70-101, Salt Lake City, Utah, 1970.

[65] C. Wylie, G. Romney, D. Evans, and A. Erdahl. Half-tone perspective drawings by computer. In *Proceedings of AFIPS 1967 FJCC*, volume 31, pages 49–58, 1967.

[66] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH 97*, pages 77–88, August 1997.

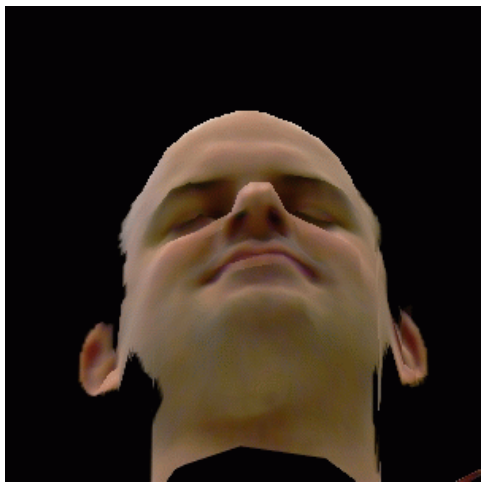
(a) Reference image

(b) Recovered depth

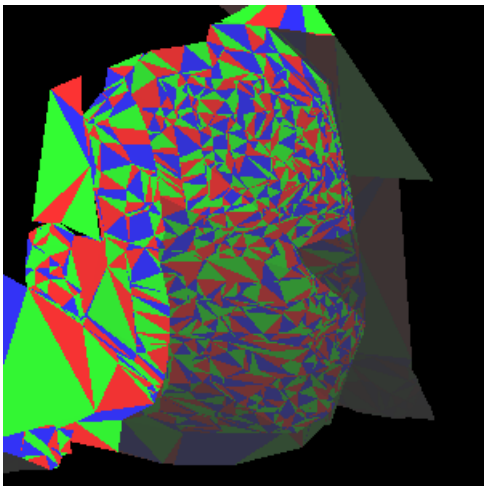


(c) BSPI

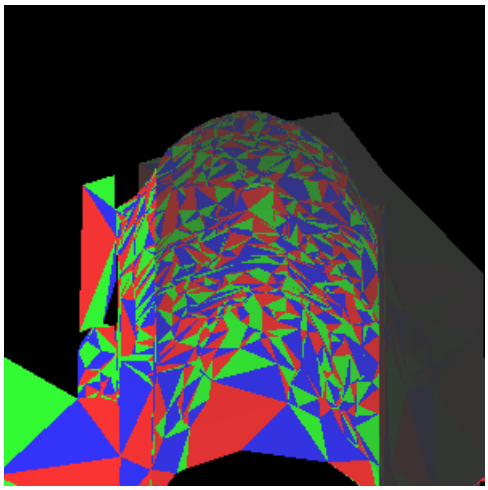
(d) Novel view



(e) Novel view



(f) Drawing order

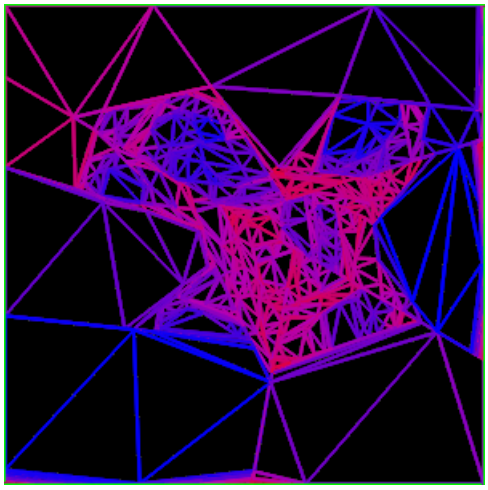


(g) Drawing order

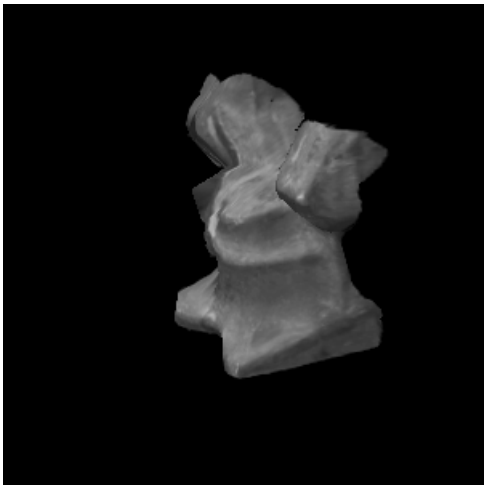
Fig. 16. Results on FACE. Average frame rate: 20.844 fps and triangle count: 2703. Input image and initial depth map courtesy of the Signal Analysis and Machine Perception Laboratory, The Ohio State University.

(a) Stereo pair

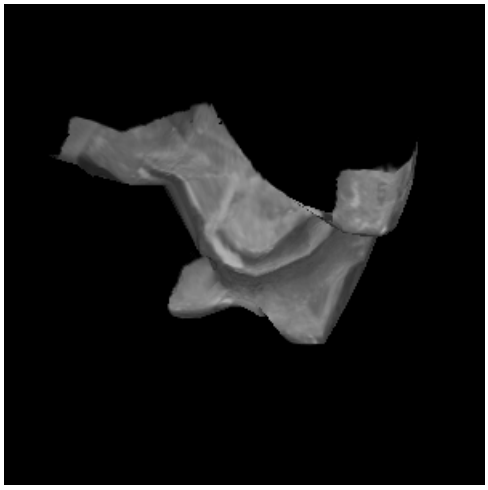
(b) Recovered depth



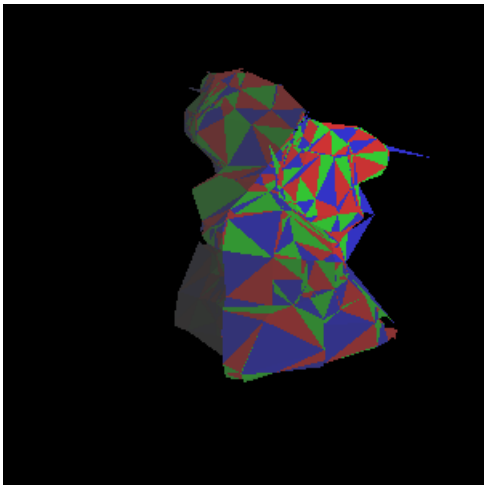
(c) BSPI



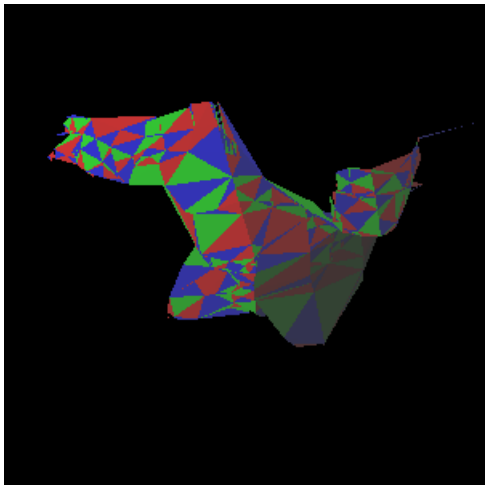
(d) Novel view



(e) Novel view

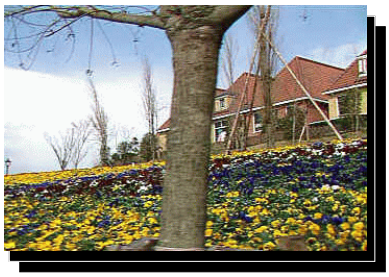


(f) Drawing order



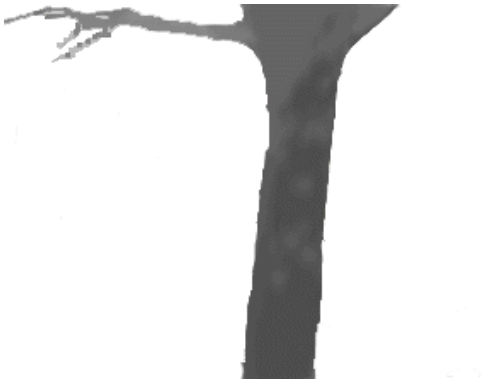
(g) Drawing order

Fig. 17. Results on RENAULT. Average frame rate: 52. 098 fps and triangle count: 1450. Input images and initial depth map courtesy of Lee and Medioni.

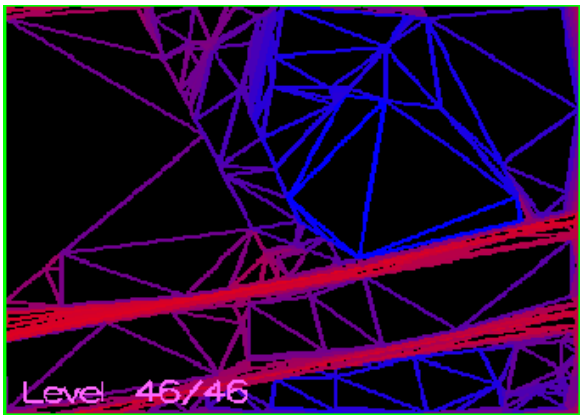


(a) Image sequence

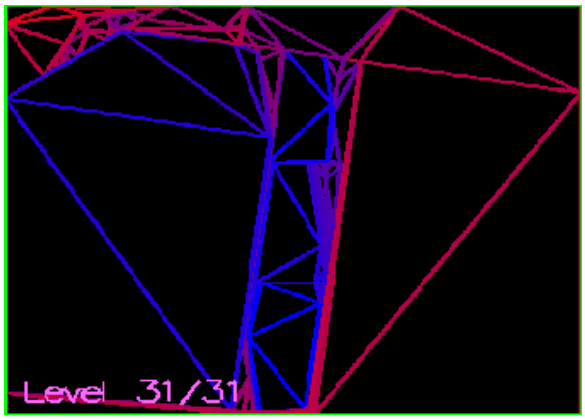
(b) Recovered background depth



(c) Recovered foreground depth



(e) Background BSPI

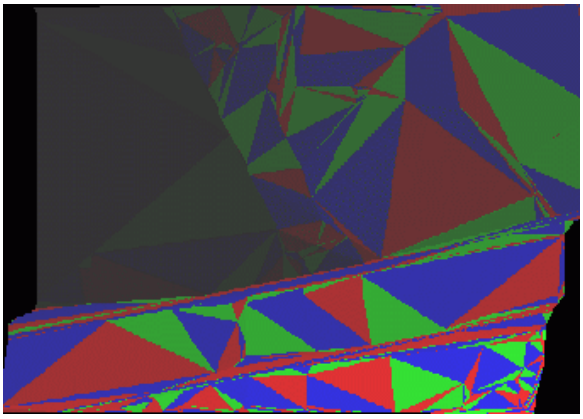


(f) Foreground BSPI

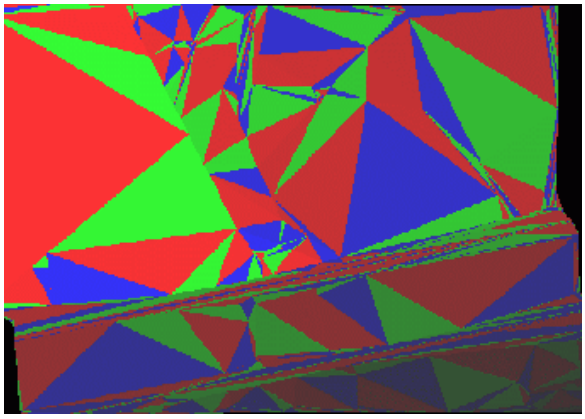
Fig. 18. Results on FLOWER GARDEN. Layered BSPI's are used to address "disocclusion." Average frame rate: 33.459 fps and triangle count: 744. Input images and initial velocity map courtesy of Gaucher and Medioni.

(g) Novel view

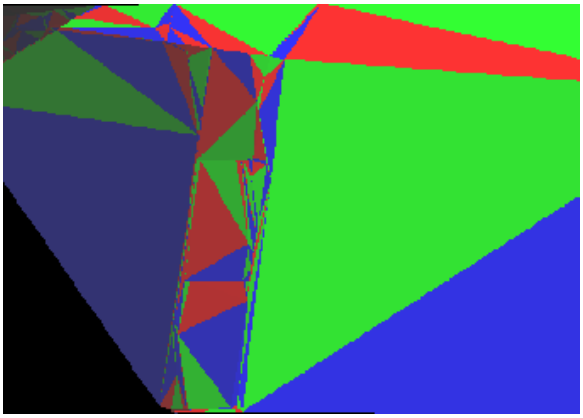
(h) Novel view



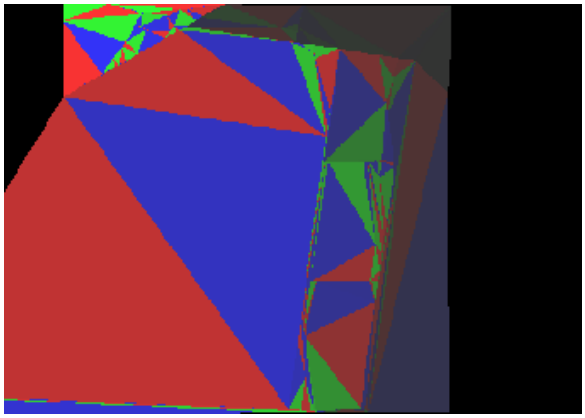
(i) Background drawing order



(j) Background drawing order



(k) Foreground drawing order



(l) Foreground drawing order

Fig. 19. Results on FLOWER GARDEN (cont'd). Average frame rate: 33.459 fps and triangle count: 744. Input images and initial velocity map courtesy of Gaucher and Medioni.