Propagation Algorithm for Image-Based Rendering Using Full or Partial Depth Information

Ha T. Nguyen, Student Member, IEEE, Minh N. Do, Member, IEEE,

Abstract—We propose a new approach, called the Propagation Algorithm, for the image-based rendering (IBR) application that synthesizes novel images, as taken by virtual cameras at arbitrary viewpoints, using a set of acquired images. The Propagation Algorithm proactively propagates all available information from actual cameras to virtual cameras, using the depth information at all or some of intensity pixels. This process turns the IBR problem into a two-dimensional interpolation problem at virtual image planes. Each virtual image thus can be efficiently rendered at once for all image pixels. Experimental results show the Propagation Algorithm produces accurate rendering, even around object boundaries — where most of the existing IBR techniques fail.

Index Terms—Image-Based Rendering, Plenoptic Function, Range Camera, Depth Information, Ray Tracing, Nonuniform Interpolation, Image Segmentation.

I. INTRODUCTION

Image-based rendering (IBR) is an emerging technology that has been developed as an alternative to traditional model-based techniques for image synthesis. IBR synthesizes novel (or virtual) images, as taken by virtual cameras at arbitrary viewpoints, using a set of acquired images. With the advantages of being photorealism and having low complexity over model-based techniques, IBR has many potential applications such as remote reality and telepresence [22], [25].

Situated at the crossroad of computer vision, computer graphics and signal processing, IBR has attracted a lot of attention in recent years. Shum et al. [22] classify IBR techniques in a continuum based on the level of prior knowledge of the scene geometry. At one end of the continuum, 3D warping [13], [14], layered-depth images (LDIs) [21], and LDI trees [2], require the full geometrical information. In view interpolation [3], view morphing [20], and joint view triangulation [11], feature correspondence between images is used as implicit geometrical information. At the other end of the continuum, light field rendering [10] and Lumigraph [7] use many images to compensate the scene geometrical information.

Despite many IBR techniques having been proposed, there is still room to improve the rendering quality and computational complexity [22]. One of the reasons is that many of the existing IBR techniques essentially follow the traditional framework, in which an intermediate step called image-based modeling reconstructs the scene geometry, explicitly or implicitly, using the acquired images. In other words, existing IBR techniques render virtual images through a reconstruction of the scene geometry, even if what we really want is not the 3D scene geometry but only the 2D virtual images. Once having the scene geometry, the ray-tracing approach is usually used to render virtual images. For each virtual pixel one draws a ray through the camera center and the virtual pixel and finds the point where this ray hits an object surface in the scene. The intensity of this point is then interpolated using information of neighboring actual pixels. The ray tracing approach is computationally complex, and has a limited rendering quality around the object boundaries [6], [24].

In this paper, we present a new approach for the IBR problem. We named this approach Propagation Algorithm [15]. Our approach differs from existing methods in two points:

1) We assume that the depth information is available for all or some image pixels (in our experiments in section IV, only about 3% of intensity pixels having depth is enough for an acceptable rendering quality).

2) We do not use the ray-tracing approach and intensity interpolation pixel by pixel on virtual images. Instead, we proactively propagate all available information to the virtual image plane first, remove occluded points in continuous domain, and then do the intensity interpolation only once for all the pixels.

For the first point, the depth assumption is widely made in literature, for both synthesized and realistic images [6], [21], [2], [13], [14], [20], [11], [3]. For synthesized images, the depth information can be computed because of the availability of the scene geometry and the camera model. For realistic images, the assumption is justified by the availability of range...
camera technology\(^1\) (like 3D range sensing [9, 17]) and by the existence of feature correspondence techniques [19] and techniques to reconstruct the scene geometry from pixel correspondences [12, 5, 8]. The second point is motivated by a fact that the IBR problem is a nonuniform interpolation problem [25]. However, our algorithm does not interpolate the generally high dimensional plenoptic function [1] or locally interpolated pixel intensities (using nearby cameras and/or nearby pixels), but it follows the classical 2D interpolation framework at the virtual image plane. This framework collects the available information first, then interpolates the intensity function at the virtual cameras at once for all pixels. The experimental results show that the Propagation Algorithm improves the rendering quality, especially around the object boundaries.

This paper is organized as follows. Section II sets up the problem of IBR with depth information. Section III describes the Propagation Algorithm and experimental results in the case where all the pixels are associated with depth information. Section IV presents the algorithm when we have the depth information for only a small subset of intensity pixels. Finally, section V presents the conclusion the paper and discussion of future work.

II. PROBLEM STATEMENT

In this paper, we consider the pinhole camera model (see Fig. 2), although the proposed algorithm should work for other camera models as well. We furthermore suppose that: i) the scene surfaces are lambertian, i.e. images of a 3D point in different cameras have the same intensity, and ii) the intensity and depth function are sufficiently smooth to be able to be interpolated from a finite set of samples.

Each camera \(C\) is specified by an intrinsic calibration matrix \(K\) and its location specified by a rotation matrix \(R\) and a translation vector \(T\) [12]. The sets \(I\) and \(D\) are the set of intensity pixels and of depth pixels at camera \(C\), respectively. Capital letter \(X\) denotes a point in the 3D world coordinate, and small letter \(x\) denotes the image of \(X\) in 2D image coordinate. The intensity and depth of \(x\) are \(I(x)\) and \(D(x)\), respectively. As the camera is supposed to be pinhole, the image formation process can be described using the following projection equation [12]:

\[
D(x) \begin{bmatrix} x \\ 1 \end{bmatrix} = [KR, KT] \begin{bmatrix} X \\ 1 \end{bmatrix}.
\]

We denote \(\vec{x} = (x, 1)^T\) and \(\hat{X} = (X, 1)^T\) be the homogeneous coordinate vector of \(x\) and \(X\), and \(\Pi = [KR, KT]\) be the projection matrix of \(C\) [12]. Note that \(\vec{x}\) and \(\hat{X}\) are defined up to a scale, that is for all \(t \in \mathbb{R}, t \neq 0\), we also have \((tx, t)^T\) and \((tX, t)^T\) in homogeneous coordinates correspond to \(x\) and \(X\) in Euclidean coordinates, respectively. In this paper, unless we specify otherwise, \(\vec{x}\) and \(\hat{X}\) have the last coordinate equal to 1. Eq. (1) then becomes a linear equation in homogeneous coordinates.

The inputs of our algorithm are calibration information \(K_i, R_i, T_i\) and \(I_i\), the intensity images \(\{I(x) : x \in I_i\}\), the depth information \(\{D(x) : x \in D_i\}\) of the actual cameras \(\{C_i\}_{i=1}^N\), and also a projection matrix \(\Pi\) and a collection of the desired pixels \(\mathcal{I}\) of the virtual camera \(C\). This implies that we need all the cameras to be calibrated in addition to the intensity and depth information. The output of the algorithm is the virtual image \(\{I(x) : x \in \mathcal{I}\}\). We consider the case we render only one virtual image. In case where collection of virtual images is desired, a natural extension of the Propagation Algorithm with some level of optimization can be implemented.

III. PROPAGATION ALGORITHM WITH FULL DEPTH INFORMATION

In this section we consider the full depth case, i.e. \(\mathcal{I}_i = D_i\) for all \(i = 1, 2, \ldots, N\). That means for every actual pixel we know both the intensity and depth information. In the first two subsections, we review the traditional ray-tracing approach and present an overview of the Propagation Algorithm to highlight its new features and differences with the traditional approach. In the next three subsections, we detail the three steps of the Propagation Algorithm: Information Propagation, Occlusion Removal and Intensity Interpolation. The last subsection presents some experimental results.

---

*Fig. 2.* The pinhole camera model. Pixel \(x\), situated at the intersection of the image plane and the ray connecting the camera center \(O\) and the 3D point \(X\), is the image of \(X\). Note that the depth \(D(x)\) is defined by the distance from \(X\) to the camera image plane.

*Fig. 3.* The inputs and output of the Propagation Algorithm for IBR. The inputs are the (intrinsic and extrinsic) calibration, intensity and depth information at the actual cameras, and the calibration information at the virtual camera. The output is the virtual image at the virtual camera.
A. Traditional ray-tracing approach

In the traditional ray-tracing approach, we render virtual images pixel by pixel. For each virtual pixel, we first try to recover the depth associated with this virtual pixel, i.e. where the ray through the camera center and the virtual pixel hits an object surface in the scene. In IBR reconstruction-based techniques, this requires the reconstruction of the scene geometry, which is very costly or even unrealistic. To avoid the geometrical reconstruction, techniques like local color consistency (comparing the intensity of corresponding pixels [24]) or depth matching (comparing the depth of corresponding pixels [6]) have been proposed to find the depth. These techniques, instead of finding the intersection of the virtual ray with the object surface, test a collection of points along the virtual ray, and return the one that produces the best consistency of intensity and/or depth. For example in Fig. 4(a), we test all points along the dash line through the center of virtual camera $u$ ($P$ is such a point.)

After having the depth or the intersection point, we trace back to the actual cameras to get the corresponding pixels. The rendered intensity is then computed using the corresponding pixels found from actual images. Ray-tracing techniques are hence computationally complex, and even with the help of depth information, the rendering quality is still limited because of depth discontinuities around object boundaries [6]. Also, because the images contain just discrete samples of the intensity function, exact pixel correspondences hardly exist. Thus, the use of approximated pixel correspondences may produce distortion of object images in the rendered images.

The differences between the ray-tracing approach and the approach of the Propagation Algorithm is shown in Fig. 4. In the ray-tracing approach, the flow of information is from the virtual cameras back to the actual cameras whenever we need the (intensity) information. On the contrary, in the Propagation Algorithm, the flow of information is from the actual cameras to the virtual camera. That is we collect all the available information from actual cameras and send them to the virtual camera before we actually do any further processing.

B. Overview of Propagation Algorithm

The key idea of the Propagation Algorithm is proactivity. Using the depth information we can proactively propagate all available information from actual cameras to the virtual camera. Then we propose a simple occlusion removal technique to keep only points relevant to the rendering of the virtual image. These steps are realized in the continuous domain at the virtual image plane, which is also a key feature differentiating the Propagation Algorithm to existing IBR methods. Finally, using remaining points after the occlusion removal step, we interpolate the intensity at once for all virtual pixels.

To illustrate the main idea, we use the simple 2D light field setting [10], [7], [6] as depicted in Fig. 7. In this setting, all the cameras lie on the camera line $d = 0$, and all the image lines are on the line $d = f$. When we say an “image” we mean a 1D signal (e.g. scan-lines in 2D images). The Propagation Algorithm contains three steps as shown in Fig. 5, illustrated in Fig. 6, and detailed in the next subsections. The following is the main idea:

1. **Information Propagation.** In this step we propagate all the available information to the virtual camera image plane. This can be done using the depth information of pixels.

   In the setting as in Fig. 7, the intensity information at pixels $p_0$ and $q_0$ of the actual camera $u_0$ is propagated to the points $p$ and $q$ of the virtual camera $u$, respectively. Fig. 6(a) shows a scene viewed from the virtual camera at $u = 4$. The circles are propagated from the actual camera at $u_0 = 2$, and the dots are propagated from the actual camera at $u_1 = 6$.

2. **Occlusion Removal.** In this step we remove all the points in whose neighborhood there is another point with noticeably
Fig. 6. Illustration of Propagation Algorithm for 2D light field example. (a): the scene’s depth viewed from virtual camera at $u = 4$ and propagated points from actual cameras at $u_0 = 2$ (circle) and $u_1 = 6$ (dot). (b): removed points (x-marker) and remaining points (dot). (c): remaining points (dot) and linearly-interpolated virtual pixels (circle) compared with the actual ones (line). Note that (c) has the intensity as vertical axis, instead of depth as (a) and (b).

smaller depth; these points are likely occluded at the virtual camera.

For example in the Fig. 7 point $p$ is propagated from the pixel $p_0$, but then it should be removed because it is occluded by the object OBJ. In Fig. 6(b), occluded points are shown in x-markers, and remaining points in dots.

3. Intensity Interpolation. In this step we interpolate the remaining information using an appropriate kernel function depending on the characteristics of the intensity function.

In Fig. 6(c) we show the remaining points in dots, the interpolated intensity values (virtual image) in circles, and the actual image with a line.

There are several advantages of this three-step framework. First, such proactive use of available information allows us to change the number of cameras during the rendering process (e.g., in a real-time system) without considerably modifying the algorithm. Second, by doing the intensity interpolation at the end, we do not calculate the intensity on a pixel by pixel basis, but rather we get the whole virtual image at once. This approach reduces the computational complexity as well as provides flexibility in the resolution of the virtual image. Third, our approach also allows us to obtain the intensity for pixels near image borders, where ray-tracing techniques fail because of the lack of pixel correspondences. Forth, we avoid image distortion at the virtual image as we consistently perform the rendering process in the continuous domain. Finally, in case of multiple virtual cameras, the Information Propagation step can be done once for all virtual cameras, before we independently render each virtual camera. This common process helps speed up the rendering process, because we reduce the data access load at actual cameras.

C. Information propagation

The idea of the Information Propagation step is that we collect all the available information to the virtual camera before actually processing the information. For each actual
The equation shows that the point from its image relative to a camera calibration information $K$, $R$, and $T$. The depth information allows us to recover the point in the 3D world coordinate. With this information, we can exactly compute the image position of the 3D point at the virtual image plane (this may not be at pixel positions) and the relative depth to the virtual camera. The intensity, depth and position relative to the virtual camera will be propagated to the virtual camera $C$ without any further processing. In Fig. 7, if point $P$ was visible at the virtual camera $u$, then its image must have been at position $p$ with the same intensity as pixel $p_0$. The corresponding depths at $p_0$ and $p$ are also the same. Note that the depth is defined as the distance of a surface point to the image plane, and not to the camera center (see Fig. 2). We say that the intensity and depth information of pixel $p_0$ of the actual camera $C_0$ are propagated to point $p$ of the virtual camera $C$.

We show now how to recover a 3D point $X$ from its image $x$ at a (virtual or actual) camera $C$ and the corresponding depth, and conversely, how to compute the depth of a 3D point $X$ relative to a camera $C$. Let us start from the projection equation (1):

$$ (1) \iff \begin{pmatrix} D(x)\bar{x} \\ 1 \end{pmatrix} = \begin{pmatrix} KR & KT \\ 0 & 1 \end{pmatrix} \bar{X} $$
$$ \iff \bar{X} = \begin{pmatrix} KR & KT \\ 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} D(x)\bar{x} \\ 1 \end{pmatrix} $$
$$ \iff \bar{X} = \begin{pmatrix} R^{-1}K^{-1} & -R^{-1}T \\ 0 & 1 \end{pmatrix} \begin{pmatrix} D(x)\bar{x} \\ 1 \end{pmatrix} $$
$$ \iff X = D(x)R^{-1}K^{-1}\bar{x} - R^{-1}T. $$

Eq. (2) is well-defined because $K$ and $R$ are invertible. This equation shows that the 3D point $X$ can be explicitly recovered from its image $\bar{x}$ if the depth $D(x)$ is known in addition to the calibration information $K$, $R$, and $T$. To compute the depth of $X$ relative to a camera $C$, we once again derive from Eq. (1). The third coordinate of (1) gives:

$$ D(x) = \pi_3(\Pi \bar{X}), $$

in which $\pi_3$ is the operator returning the third coordinate of a vector in $\mathbb{R}^3$: $\pi_3((a, b, c)^T) = c$.

Fig. 8 summarizes the Information Propagation step using pseudocode. This step can be done for each actual camera independently. So our algorithm can work with an arbitrary number of actual cameras. This number can be modified anytime during the running of the algorithm without modifying the algorithm. Note that at this point we still work in the continuous domain and do not pixelize the image position.

In practice, some depth error may result due to the processing of the depth image (for example using a lowpass filter). Around the boundaries, this error is usually high compared to the error due to the quantization process or depth measurement. Thus, we may find some outliers after the Information Propagation step. Those are points pushed very far away from their actual positions. This can cause other points around outliers with larger depth to be removed in the next step. To overcome this problem, we detect these situations by, for example, comparing the intensity of points with their neighbors. A high change of intensity value in all directions can be a good sign of outliers, contrary to changes in some direction at image edges.

### D. Occlusion removal:

In Fig. 7 the pixel $p_0$ is propagated to the point $p$, though the surface point $P$ is occluded by the object OBJ. This subsection presents how we can prevent occluded points like $p$ from being taken into account in image rendering. The main idea of this step is based on the following observation. If a 3D point is occluded at the virtual camera, such as $P$ in Fig. 7, then there should be another point, such as $Q$ in Fig. 7, in the neighborhood of $P$ with a smaller depth.

In the Occlusion Removal step, we remove all the points in whose $\epsilon$-distance neighborhood there exists at least another point with $\sigma$-smaller depth. Fig. 9 illustrates the main idea of the Occlusion Removal step: when we consider point $A$, we create a removal zone (the shaded zone) for which all other points falling in this zone will be removed. This process is performed for all points after the Information Propagation step. Fig. 10 shows the pseudocode of this step.

The parameters $\sigma$ and $\epsilon$ can be tuned to be appropriate with the characteristics of the scene and/or applications. A larger $\sigma$ tends to keep more points, especially at object surface where the depth changes fast, though this may be done at the risk of including points from hidden objects. A larger $\epsilon$ removes

---

**Fig. 7.** A simple 2D light field setting with occlusion. In this setting, an “image” is a line on the image focal line.

**Fig. 8.** Pseudocode of the Information Propagation step. All actual pixels are propagated from actual cameras $\{C_i\}_{i=1}^N$ to virtual camera $C$ using the depth information. No further processing is needed at this point.
looking at point A occluded by point A.

Fig. 9. Illustration of the Occlusion Removal step. Point C is considered occluded by point A, and therefore is removed. Point B is not removed by looking at point A alone.

more points, and keeps only points with high confidence to be visible, though the price is that the image quality, especially around boundaries, may be reduced.

E. Intensity Interpolation

The Intensity Interpolation step is to render the virtual image using the remaining points Q after the Occlusion Removal step, that is when all the information relevant to the rendering of the virtual image is at our disposal. As we assume that the intensity function is smooth enough, we can interpolate the intensity function using the intensity information of points in Q. The virtual image is then simply the value of this function at the pixel positions (the pixelization process). Fig. 6(c) illustrates the main idea of the Intensity Interpolation step, and Fig. 11 shows the pseudocode of the step.

One issue of the Intensity Interpolation step is the choice of the kernel function. In the case of full depth, the remaining points Q are usually very dense at the virtual camera image plane. In practice, a simple tool like linear interpolation can provide good rendering quality with a small computational cost, as we use in our experiments.

There is a notable difference between 1D nonuniform interpolation and 2D nonuniform interpolation. In 1D interpolation, the presence of a total order in \( \mathbb{R} \) allows us to interpolate the function values using relevant samples, for example to linearly interpolate consecutive samples in linear interpolation. In the 2D case, there is no total order in \( \mathbb{R}^2 \). We need first to partition the plane \( \mathbb{R}^2 \) using some triangulation before performing the interpolation. For example, in the case of 2D linear interpolation, one usually uses the Delaunay triangulation to partition the plane, then for each triangle of the triangulation, we interpolate the function values linearly using the values at three triangle vertices.

F. Experimental results

In our experiments, we use the stereo data acquired by Scharstein et al. [18]. In this data, the disparity maps are given instead of the depth, though one can be obtained easily from the other.

The configuration used in the experiments is purely translational. We have two actual cameras at \( u_0 = 2 \) and \( u_1 = 6 \), and each one associates with an intensity image and a disparity map. The actual intensity image at \( u = 4 \) is also given, and is used as ground truth to evaluate the rendered image of a virtual camera at \( u = 4 \). Because the images are color, we render the virtual image for each color channel (RGB) separately.

Fig. 13 demonstrates our result for Teddy images using parameter values \( \epsilon = 0.6 \) and \( \sigma = 0.05D(y) \) in the Occlusion Removal step, and linear interpolation in the Intensity Interpolation step. We can see that the proposed algorithm produces a very good result, even around boundaries where most existing IBR techniques fail.

IV. PROPAGATION ALGORITHM WITH PARTIAL DEPTH INFORMATION

The Full Depth Propagation Algorithm relies on the availability of depth information. The more depth information we have, the more dense the propagated points at the virtual image plane are, and so the better rendering quality we can expect. However, in practice, we do not always have the full depth information at our disposal. The current range camera technology has a lower resolution than the intensity camera technology [9], [16]. If the depth is obtained by correspondence techniques [19], we only have the depth at pixels at easy-to-handle features (such as edges and corners). Thus, an algorithm for IBR using partial depth information is important to investigate.

While the feature correspondence problem remains as one of the main bottlenecks in the field of computer vision, the range camera technology is an emerging alternative for

\[ \% \text{Interpolate the intensity function} \]

\[ I_{\text{interpolate}} = \text{Interpolate}(Q, \{I(x) : x \in Q\}, \text{linear}); \]

\[ \% \text{Pixelize } I_{\text{interpolate}} \text{ to get the virtual image} \]

\[ \text{For each } x \in I \]

\[ I(x) = I_{\text{interpolate}}(x); \]

\[ \text{Endfor} \]

Fig. 11. Pseudocode of the Intensity Interpolation step. We interpolate the intensity function using remaining points after the Occlusion Removal step. The virtual image is then obtained by a pixelization of the intensity function.

\[ \text{The data is available at } \text{http://cat.middlebury.edu/stereo/newdata.html}. \]
Fig. 12. Input images of Full Depth Propagation Algorithm. (a): intensity image at $u_0 = 2$, (b): intensity image at $u_1 = 6$, (c): depth image at $u_0 = 2$, and (d): depth image at $u_1 = 6$.

Fig. 13. Rendered image of the Full Depth Propagation Algorithm. (a): the ground truth image taken at $u = 4$, and (b): the rendered image at $u = 4$. 
Fig. 14. The scene at $u_0 = 2$ and depth available pixels (highlighted). The depth is available on a regular grid (dots). In our experiments, the grid is 6 intensity pixel distance in both directions (about 3% of number of intensity pixels).

stereo applications. It provides more accurate and detailed information of the scene geometry. Hence, we decide to focus on the setting where the depth information is obtained by range cameras in this paper, and leave the case of partial depth IBR with nonuniform depth samples for future research. That is, in this section, we consider the case where the depth information is available at a uniform grid of pixels, or a downsampled version of the full depth used in section III.

Fig. 14 highlights the pixels with depth available. In our experiments, the depth is available in a grid of $k = 6$ intensity pixel distance in both vertical and horizontal directions. In this setting, the amount of depth information is about 3% of the total number of pixels.

A. Preprocessing or postprocessing

As the Full Depth Propagation Algorithm works well as shown in Section III, there are naturally two approaches to further continue in the direction of solving the IBR problem using partial depth information: preprocessing and postprocessing. In the preprocessing approach, we interpolate the depth function first, then provide it to the Information Propagation step of the Full Depth Propagation Algorithm. In the postprocessing approach, we just propagate the depth-available pixels, remove occluded points, then interpolate the intensity function.

Among these two approaches, we adopt the preprocessing one for several reasons. First, the depth is much more regular than the intensity to handle (see Fig. 15), since the object surfaces of the 3D scene can be approximated by an union of planar pieces. Thus, simple techniques such as bilinear interpolation and nearest neighbor interpolation work well to interpolate the depth. Furthermore, the number of discontinuities is usually smaller in depth images than in intensity images. The second reason is that this approach makes use of intensity information at depth-unavailable pixels, while the postprocessing approach simply discards this information.

B. Nearest neighbor interpolation and bilinear interpolation of the depth image

To interpolate the depth images from the samples available at an uniform grid, there are two simple ways to approximate the depth in 2D: nearest neighbor interpolation and bilinear interpolation. Nearest neighbor interpolation uses the depth value of the nearest depth-available point to interpolate the depth function. This technique is similar to the piecewise constant interpolation in one dimension. It approximates decently the regions around the object boundaries.

Bilinear interpolation, for each sample position, linearly interpolates the value of the depth function at four samples of the square containing the sample in question. It is equivalent to a concatenation of two linear interpolations in both vertical and horizontal directions. This is similar to the linear interpolation in classical 1D interpolation. It works best on the object surfaces, which can be approximated as union of planar pieces on the 3D scene.

Fig. 16 shows the rendered images with the depth is interpolated using nearest neighbor interpolation and bilinear interpolation in the preprocessing step. Although the rendering quality is acceptable on object surfaces, we can notice artifacts around the object boundaries. Nearest neighbor interpolation produces blocky edges while bilinear interpolation results in blurred edges.

C. Segment-wise depth interpolation

Though both techniques sound simplistic, they can produce decent results (see Fig. 16). Nevertheless, the rendering quality around the object boundaries is still limited. Nearest neighbor interpolation and bilinear interpolation do not make full use of the knowledge of the intensity images. There is a similarity between the intensity value of neighboring pixels in images that is not efficiently exploited.

In order to improve the rendering quality, in this subsection we propose a new technique, called segment-wise depth interpolation, to incorporate both intensity and depth information.
Nearest neighbor interpolation of depth.

Bilinear interpolation of depth.

Fig. 16. Rendered images using simple techniques to interpolate the depth. (a): the rendered image using nearest neighbor interpolation of depth, and (b): the rendered image using bilinear interpolation of depth. The distortion of the rendered image around the boundaries is still noticeable (blocky edges for nearest neighbor interpolation and blurred edges for bilinear interpolation). See Fig. 20 for the rendered image of the proposed Partial Depth Propagation Algorithm.

Only intensity available

Depth is also available

Unit square S

Pixel x to interpolate the depth

Segmentation boundaries

Fig. 17. Pixel positions at an actual camera image plane. Points are intensity pixels, circles are pixels with depth information in addition to the intensity information. The downsample rate in both directions in this example is $k = 6$ as in our experiments. The boxed square is an example illustrating how we divide the image plane into squares to interpolate the depth in the algorithm.

available to approximate the depth. The proposed technique uses an image segmentation technique to segment the image into different regions. The idea is that the depth image will be interpolated using only depth-available pixels in the same object surface.

To begin, we use an image segmentation technique to obtain a segmentation of the depth images. We will interpolate the depth square by square, each square is an unit square of the depth image, as illustrated in Fig. 17. For each unit square $S$, if all of its four vertices belong to a same depth segment, we use bilinear interpolation technique to interpolate the depth of all other pixels inside $S$. Otherwise, $S$ lies on boundaries of different depth segments (so likely on different object surfaces). For each particular pixel $x$ inside $S$, we determine which vertices of $S$ belong to the same intensity segment with $x$. The nearest neighbor interpolation technique is used to approximate the depth of $x$ using these vertices. Fig. 18 shows the pseudocode of the proposed segment-wise depth interpolation.

In practice, some pixels belonging to tiny intensity segments (usually segments of texture on object surfaces) may be found to not sharing segments with any vertex of $S$. These pixels are marked and filled with the depth value of their neighbors at the end, using for example morphological imaging techniques [23]. Some pixels that lie in the interval between two depth-available pixels can be classified to either unit square they belong to.

Note that approximating the depth is much less complex than trying to reconstruct the scene geometry, because we only deal with 2D intensity and depth images, instead of 3D objects. Fig. 19 shows the reconstructed depth of the proposed technique for the case where depth images are downsampled versions of the intensity images with rate $k = 6$ for both directions. That is we need only one depth pixel every $k^2 = 36$ intensity pixels, or equivalently, we need the depth information at less than 3% of the number of intensity pixels.

D. Experimental results

In our experiments, we use the image segmentation technique called Mean Shift proposed by Comaniciu et. al. [4], and its software EDISON.

Fig. 19 shows the reconstructed depth images using the segment-wise interpolation technique. These depth images are then provided to the Information Propagation step of the Full Depth Propagation Algorithm to render the virtual

3The software is available at http://www.caip.rutgers.edu/riul/research/code/EDISON/.
% Segmentation of intensity and depth images

\[ \text{segm}_i = \text{Segmentation}(I(x) : x \in I) \]
\[ \text{segm}_d = \text{Segmentation}(D(x) : x \in D) \]

For each unit square \( S \) of depth image \( D(x) \)
If all the vertices of \( S \) belong to the same segment of \( \text{segm}_d \)
\[ \mathcal{V} = \text{Set of vertices of } S \]
\[ D(x)|_S = \text{Interpolate}(\mathcal{V}, \{D(x) : x \in \mathcal{V}\}, \text{bilinear}) \]
Else
\[ \mathcal{V} = \text{Set of vertices of } S \text{ sharing the same } \text{segm}_i \text{ segment with } x \]
\[ D(x) = \text{Interpolate}(\mathcal{V}, D(x)|_\mathcal{V}, \text{nearest}) \]
Endif
Endfor

Fig. 18. Pseudocode of the Segment-wise interpolation of depth. The depth is interpolated for each unit square of the depth image. Linear interpolation is used on object surfaces (segments of depth image). Nearest neighbor interpolation is used around the object boundaries.

(a) Interpolated depth at \( u_0 = 2 \).
(b) Interpolated depth at \( u_1 = 6 \).

Fig. 19. The reconstructed depth using segment-wise depth interpolation technique. (a): interpolated depth at \( u_0 = 2 \), and (b): interpolated depth at \( u_1 = 6 \).

image. Fig. 20 shows the rendered image of the Partial Depth Propagation Algorithm using about 3\% of depth (\( k = 6 \)), together with the rendered image using the Full Depth Propagation Algorithm and the ground truth image. We note that the Partial Depth Propagation Algorithm does improve the rendering quality around object boundaries compared to the nearest neighbor depth interpolation and bilinear depth interpolation. Its rendering quality is even comparable to the rendering quality of the Full Depth Propagation Algorithm.

V. CONCLUSION AND DISCUSSION

In this paper we present a new approach, called the Propagation Algorithm, for the IBR problem. The algorithm includes three steps: Information Propagation, Occlusion Removal and Intensity Interpolation. By assuming the availability of the depth information at all or some image pixels, the algorithm produces an excellent improvement of rendering quality, especially around the object boundaries. Moreover, the approach provides a framework for a rigorous and systematic investigation of the rendering quality by turning the IBR problem from a 7\( D \) interpolation problem of the plenoptic function into a classical 2\( D \) interpolation problem.

The proposed Propagation Algorithm turns the IBR problem into a classical nonuniform sampling and interpolation problem at the virtual image plane. Moreover, the dimension of the problem is considerably reduced to 2\( D \) (from 7\( D \) of the general plenoptic function [1] or 4\( D \) in light field setting [10]). In some practical cases, such as that of purely translational configuration, we can even reduce the dimension of the problem to 1\( D \) interpolation by rendering the virtual images by scan-lines.

The approach also opens new room to bring more signal processing flavor into the IBR problem. An error of intensity results in the sample error of the intensity function, and an error of the depth results in a jiter at the sample positions. Thus, we can analyze the rendering error in a more systematic and rigorous way.

Finally, as we already discussed in Section IV, the case where the depth is available at a nonuniform set is left for future research. This is the case where depth is recovered
from stereo reconstruction techniques using pixel correspondences [12], [5], [8].

REFERENCES

Fig. 20. Virtual images at $u = 4$ rendered by Propagation Algorithm. (a): ground truth *Teddy* image at $u = 4$, (b): rendered image using Full Depth Propagation Algorithm, and (c): rendered image using about 3% of depth rendered by Partial Depth Propagation Algorithm.

Fig. 21. Virtual images at $u = 4$ rendered by Propagation Algorithm. (a): ground truth *Cones* image at $u = 4$, (b): rendered image using Full Depth Propagation Algorithm, and (c): rendered image using about 3% of depth rendered by Partial Depth Propagation Algorithm.