Active Rearranged Capturing of Image-Based Rendering Scenes — Theory and Practice

Cha Zhang, *Member, IEEE,* and Tsuhan Chen, *Senior Member, IEEE*

Abstract

In this paper, we propose to capture image-based rendering scenes using a novel approach called active rearranged capturing (ARC). Given the total number of available cameras, ARC moves them strategically on the camera plane in order to minimize the sum of squared rendering errors for a given set of light rays to be rendered. Assuming the scene changes slowly so that the optimized camera locations are valid in the next time instance, we formulate the problem as a recursive weighted vector quantization problem, which can be solved efficiently. The ARC approach is verified on both synthetic and real-world scenes. In particular, a large self-reconfigurable camera array is built to demonstrate ARC's performance on real-world scenes. The system renders virtual views at 5-10 frames per second depending on the scene complexity on a moderately equipped computer. Given the virtual view point, the cameras move on a set of rails to perform ARC and improve the rendering quality on the fly.

Index Terms

Active sampling, active rearranged capturing, image-based rendering, camera array, self-reconfigurable.

Manuscript received xxx xx, 2005; revised xxx xx, 200x. Work performed at CMU and supported by NSF Career Award 9984858.

C. Zhang is with Microsoft Research, Redmond; T. Chen is with Department of Electrical and Computer Engineering, Carnegie Mellon University.

Active Rearranged Capturing of Image-Based Rendering Scenes — Theory and Practice

I. Introduction

Image-based rendering (IBR) has attracted much attention recently [1], [2]. By capturing a dense set of images of a scene, IBR is capable of rendering 3D novel views with little or no scene geometry. Compared with traditional model-based rendering methods, IBR bypasses the most difficult geometry reconstruction stage, and is easy to capture and fast to render.

While geometric models are often considered the most compact representation for real-world scenes, images are very redundant. In IBR, in order to synthesize novel views directly from light ray interpolation, thousands or millions of images need to be taken from nearby view points, making them difficult to store and manage. The following question thus becomes very important for IBR: *How many images are necessary for rendering high-quality images of an IBR scene?* This problem is widely referred as the IBR sampling problem or plenoptic sampling problem, and was addressed by work in [3], [4]. In short, the plenoptic sampling problem is a multi-dimensional signal sampling problem. If the light rays are sampled uniformly in the space, the sampling rate in order to achieve interpolation without aliasing is determined by the range of depths of the scene, the surface property of the scene objects, and the occlusions between them [4]. The sampling rate is also related to the amount of information we know about the scene geometry, and at what resolution the rendering will be carried out [3]. The plenoptic sampling theory is deeply grounded on the multi-dimensional signal processing framework, and greatly enhances our understanding of IBR from a signal processing point of view.

Unfortunately, the above plenoptic sampling theory does not provide a practical guidance for capturing IBR scenes. In order to estimate the minimum sampling rate, one has to know the scene well, including its depth variation, the object surface property, and occlusions between objects. These characteristics are often difficult to measure before capturing. To overcome the limitation, Zhang and Chen [5] proposed to sample the scene with a high sampling rate, and then throw away samples by analyzing the Fourier spectrum of the over-sampled data. After the down-sampling process, the amount of data stored is minimized according to the sampling theory. The drawback is that the oversampling stage is often too expensive to perform due to time or storage constraints.

One practical approach to capturing IBR scenes is through *active sampling*, more accurately, *active incremental sampling* [6], [7]. The idea is to start with a sparse set of images of the scene, and gradually add more samples or take more images of the scene from where the rendering quality is not good. Here the rendering quality is estimated by measuring the local color consistency score [6], which has been proved to be a good indicator of the final rendering quality. In active incremental sampling, we assume once an image is taken, it will not be discarded, and as the sampling proceeds, potentially infinite number of images can be taken for the scene. Obviously, active incremental sampling is useful for capturing static scenes.

Recently there has been increasing interest in capturing dynamic scenes with IBR [8], [9]. As the scene is changing, we can no longer use a single camera to capture it. Instead, tens or hundreds of cameras need to be performing the capturing task simultaneously. This brings new challenges to the IBR sampling problem. Namely, *if the number of images one can take is limited, where shall we capture these images?* In this paper, we will address this problem with an approach we call *active rearranged capturing* (ARC). Based on what the cameras saw in the previous instance and what views need to be synthesized, active rearranged capturing re-positions the cameras so that the quality of a given set of rendered views is optimized. We formulate the problem as a recursive weighted vector quantization (VQ) problem, which can be solved efficiently.

This paper goes beyond the theoretical analysis and also presents a self-reconfigurable camera array that verifies the above ARC theory. The system is composed of 48 cameras mounted on mobile platforms. These cameras are controlled by a single computer, which performs image decoding, camera calibration, lens distortion correction, rough scene geometry reconstruction and novel view synthesis, all in real time (5-10 fps). More importantly, we utilize a simplified ARC algorithm and demonstrate that by reconfiguring the camera locations on the platform, the rendering quality of the virtual views can be significantly improved.

The paper is organized as follows. Related work is presented in Section II. The active rearranged sampling process is described in Section III. Section IV shows our self-reconfigurable camera array and how the active rearranged sampling theory is used in practice. Conclusions are given in Section V.

II. Related work

A. IBR Scene Capturing and Rendering

We first give a brief introduction on IBR scene capturing and rendering. Fig. 1 shows a typical IBR system. We place a set of cameras around the object and shoot images. In the light field [10] setup, the cameras are uniformly distributed on a plane (namely the camera plane), and point to the same direction. In the concentric mosaics [11], the cameras are arranged along a circle. In the most general form, the cameras can be anywhere, which can still be rendered through the unstructured Lumigraph rendering [12]. In this paper, we assume that the cameras are constrained on a camera plane, but on that plane the distribution of the cameras can be non-uniform. The directions of the cameras are assumed to be the same, although this requirement is not crucial to the proposed algorithm.

To render novel views from the captured images, we split them into many light rays and obtain their intensities one by one. As shown in Fig. 1, consider one of the light rays being rendered. We first trace the light ray back to the scene geometry, and obtain the crossing point O. Since the geometry is typically unknown for real-world scenes, people usually assume a constant depth plane. We then project O to the neighboring captured images (circled by an ellipse in Fig. 1) and obtain the light ray's intensity through weighted interpolation of all the projected pixels. The weights are usually determined by the angular differences between the rendered light ray and the projection directions from O to the captured images, which are also used to select neighboring images. The smaller the angular difference, the higher the weight to the associated image. Other factors such as resolution or field of view may also affect the weights [12], however they are not considered in this paper.

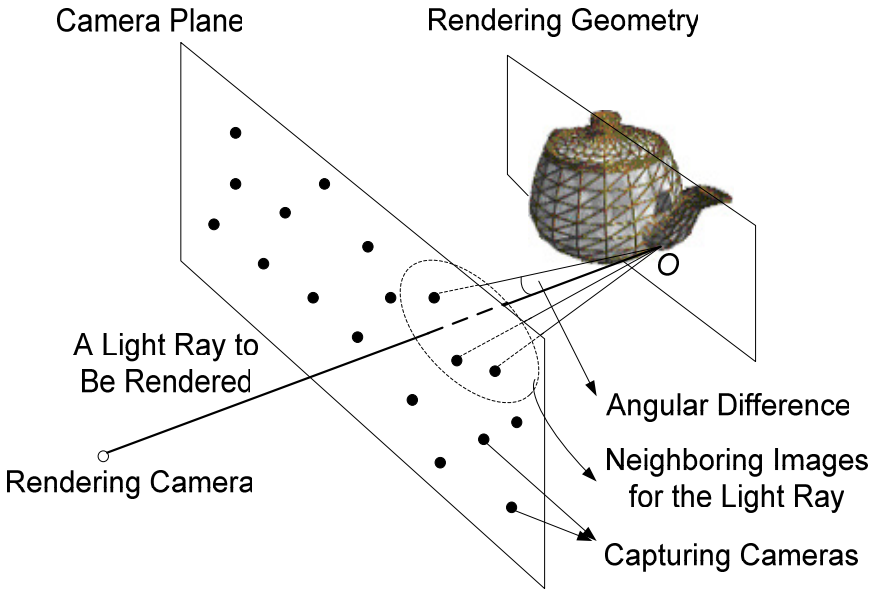

Fig. 1. The capturing and rendering of IBR scenes.

Image-based rendering covers a huge research field, and we refer the reader to [1], [2] for a detailed survey on this topic.

B. IBR Sampling

Early work on IBR sampling focused on the uniform sampling analysis of IBR. Work by Lin and Shum [13], Chai et al. [3], as well as Zhang and Chen [4] all greatly enhanced the understanding of IBR from the signal processing's point of view. In this section, we review some of the nonuniform sampling work for IBR, as they are more closely related to what we propose in this paper.

Fleishman et al. [14] proposed an automatic camera placement algorithm for IBR. A mesh model of the scene is known. The goal is to place the cameras optimally such that the captured images can form the best texture map for the mesh model. They found that such problem can be regarded as a 3D art gallery problem, which is NP-hard [15]. They then proposed an approximation solution for the problem by testing a large set of camera positions and selecting the ones with higher gain rank. The gain was defined based on the portion of the image that can be used for the texture map. A similar approach was proposed in [16], where the set of reference views were selected from a large image pool in order to minimize a certain target function. In [17], Namboori et al. developed an adaptive sample reduction algorithm for layered depth image, which is another representation of IBR.

Schirmacher et al. [18] proposed an adaptive acquisition scheme for a light field setup. Assuming the scene geometry is known, they added cameras recursively on the camera plane by predicting the potential improvement in rendering quality when adding a certain view. This a-priori error estimator accounts for both visibility problems and illumination effects such as specular highlights to some extent. A similar approach was proposed by Zhang

and Chen for concentric mosaics setup in [7], where a real system was built to demonstrate the idea.

The previous approaches mentioned above are examples of sample reduction or active incremental sampling. The active rearranged capturing concept is new, and was only very briefly mentioned in our previous paper [19] with very preliminary results. In this paper, we will re-formulate the problem in a more rigorous way, propose a weighted vector quantization based algorithm to solve the problem, and show the effectiveness of the algorithm with a self-reconfigurable camera array.

C. Existing Camera Arrays

In recent years there has been increasing interest in building camera arrays. For instance, Schirmacher et al. [18] built a 6-camera system which was composed of 3 stereo pairs and claimed that the depth could be recovered on-the-fly. The novel views are then synthesized using these depth maps. Naemura et al. [20] constructed a camera array system consisting of 16 cameras. A single depth map was reconstructed from 9 of the 16 images using a stereo matching PCI board, and was then used for rendering. Matusik et al. [21] used 4 cameras for IBR using the image-based visual hull (IBVH) technique. The IBVH algorithm was later extended to image based photo hull in [22], which also used a 4-camera array. More recently, Yang et al. [23] built a 5-camera system for real-time rendering with the help of modern graphics hardware.

Several large arrays consisting of tens of cameras have also been built, such as the Stanford multi-camera array [8], the MIT distributed light field camera [9] and the CMU 3D room [24]. These three systems have 128, 64 and 49 cameras, respectively. The Stanford system focused on grabbing synchronized video sequences onto hard drives. They also explored various applications of such a camera array, such as high dynamic video, high speed video, hybrid aperture photography, etc [25]. The CMU 3D room was able to generate good-quality novel views both spatially and temporarily [26]. It utilized the scene geometry reconstructed from a scene flow algorithm that took several minutes to run. While this is affordable for off-line processing, it cannot be used to render scenes on-the-fly. The MIT system did render live views at a high frame rate. Their method assumed constant depth of the scene, however, and suffered from severe ghosting artifacts due to the lack of scene geometry. Such artifacts are expected according to the plenoptic sampling analysis [3], [4].

In this paper we will present a self-reconfigurable camera array that consists of 48 cameras. Compared with the previous camera arrays, a unique characteristic of our camera array is that the cameras are mobile. A central computer controls where the cameras should move, based on our active rearranged capturing theory, in order to render the best quality images.

III. Active Rearranged Capturing

A. Problem Statement

Assume we have N cameras to capture a static or slowly-moving scene. The cameras can move freely on the camera plane, and point to the same direction. The field of view of each camera is assumed to be wide enough to cover the interested objects. During the capturing, we also have P viewers who are watching the scene. These

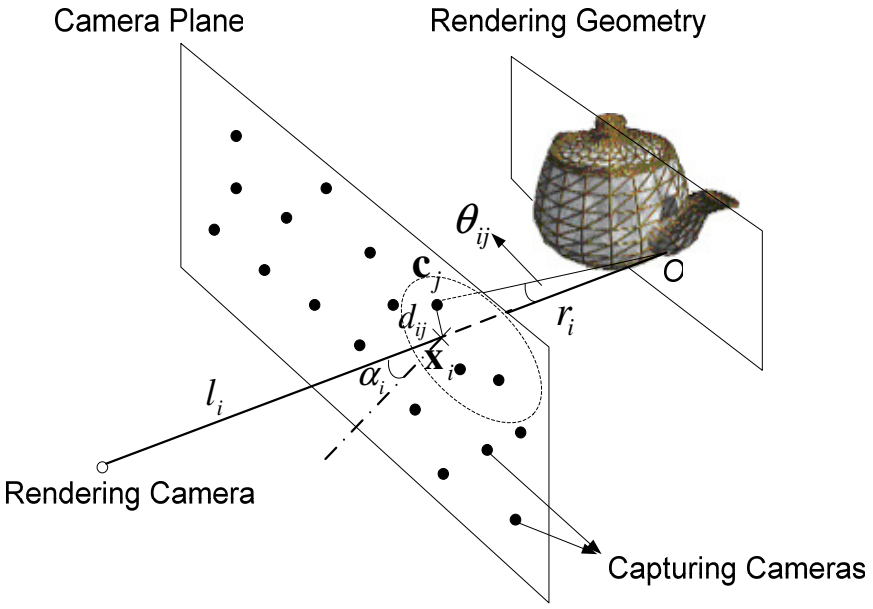

Fig. 2. The formulation of the problem.

P views are rendered through the method mentioned in Section II-A from the *N* captured images. The goal is to arrange these *N* cameras such that the *P* views can be rendered at their best quality. Here both *N* and *P* are finite.

Formally, as shown in Fig. 2, let the cameras' positions on the camera plane be $\mathbf{c}_j, j = 1, 2, \cdots, N$. Since during the rendering, each rendered view will be split into a set of light rays, we combine the *P* virtual views into *L* light rays in total. Denote them as $l_i, i = 1, 2, \cdots, L$. For a number of reasons such as insufficient sampling and inaccurate geometric modeling, the rendered light rays are not perfect. Denote the rendering errors of the light rays as $e_i, i = 1, 2, \cdots, L$, or, in vector form, $\mathbf{e}$. In our application, the rendering error can be considered as the difference between the rendered light ray and the actual light ray intensity, although other definitions are possible. Obviously, $\mathbf{e}$ depends on the camera locations $\mathbf{c}_j$. The *camera rearrangement problem* is stated as:

Definition 3.1: Given light rays $l_i, i = 1, 2, \cdots, L$ to be rendered, find the optimal camera locations $\hat{\mathbf{c}}_j, j = 1, 2, \cdots, N$, such that a function $\Psi(\mathbf{e})$ over the rendering errors is minimized. That is:

$$\hat{\mathbf{c}}_j = \arg\min_{\mathbf{c}_j} \Psi(\mathbf{e}) \tag{1}$$

The definition of function $\Psi(\mathbf{e})$ depends on the particular application. In this paper, we focus on the widely adopted squared error criterion, namely:

$$\Psi(\mathbf{e}) = \sum_{i=1}^{L} e_i^2; \tag{2}$$

We will show, in the next subsections, that such an error function leads to a weighted vector quantization solution for active rearranged capturing, which can be solved efficiently using a modified LBG-VQ algorithm [27].

Note the above camera rearrangement problem is trivial if all the *P* virtual views are on the camera plane, and $P \leq N$, because one may simply move the cameras to the virtual view positions and capture the scene at those

places directly, leading to zero error for the rendered light rays. However, the problem is non-trivial as long as one of the virtual viewpoints are out of the camera plane (even if $P = 1$), or $P > N$, because any out of plane view will have to be synthesized from multiple images, which can be potentially improved by rearranging the capturing cameras.

B. Formulation Based on the Angular Difference

In Fig. 2, denote the intersection of the light rays and the camera plane as $\mathbf{x}_i, i = 1, 2, \cdots, L$. Consider a certain light ray l_i, which crosses the scene geometry at O, and one of its neighboring camera $\mathbf{c}_j$. Denote the distance between $\mathbf{c}_j$ and $\mathbf{x}_i$ as $d_{ij} = \|\mathbf{x}_i - \mathbf{c}_j\|$ and the angular difference as θ_{ij}. Let the distance between O and $\mathbf{x}_i$ be r_i, which is known during the rendering. From the figure, we know that when the scene depth $r_i \gg d_{ij}$ (which is almost always true), we have:

$$\theta_{ij} \approx \frac{d_{ij} \cos \alpha_i}{r_i} = w_i \|\mathbf{x}_i - \mathbf{c}_j\| \tag{3}$$

where α_i is the angle between the light ray l_i and the normal of the camera plane, and $w_i = \frac{\cos \alpha_i}{r_i}$. Let

$$\widetilde{\theta}_i = \min_{j=1,\cdots,N} \theta_{ij} \tag{4}$$

be the minimum angular difference between light ray l_i and all the capturing cameras. Intuitively, if the minimum angle $\widetilde{\theta}_i$ is very small, the rendered light ray will be almost aligned with a captured light ray, hence the rendering quality should be high, even if the scene geometry is inaccurate or the scene surface is non-Lambertian. The relationship between the rendering error e_i and $\widetilde{\theta}_i$, however, is very complex for practical scenes due to various factors such as geometry accuracy and scene surface property. As a very coarse approximation, we assume:

$$e_i = \varepsilon_i(\widetilde{\theta}_i) \approx k_i \widetilde{\theta}_i, \tag{5}$$

where k_i is a scaling factor. The right side of Equ. 5 is indeed a linear approximation of $\varepsilon_i(\widetilde{\theta}_i)$, which is valid when $\widetilde{\theta}_i$ is very small. The scaling factor k_i, however, differs from light ray to light ray, and is generally unknown.

Active rearranged capturing then minimizes the summation of squared errors as:

$$\begin{aligned} \hat{\mathbf{c}}_j &= \arg\min_{\mathbf{c}_j} \Psi(\mathbf{e}) = \arg\min_{\mathbf{c}_j} \sum_{i=1}^{L} e_i^2 \\ &\approx \arg\min_{\mathbf{c}_j} \sum_{i=1}^{L} (k_i \widetilde{\theta}_i)^2 \\ &= \arg\min_{\mathbf{c}_j} \sum_{i=1}^{L} \gamma_i \min_{j=1,\cdots,N} \|\mathbf{x}_i - \mathbf{c}_j\|^2, \end{aligned} \tag{6}$$

where $\gamma_i = w_i^2 k_i^2$. Note the last equality is due to Equ. 3 and 4. The above formulation is a standard weighted vector quantization problem, and can be easily solved if the weights γ_i are known. Unfortunately, as mentioned earlier, k_i depends on the geometry accuracy and scene surface property, which is generally unknown.

Although Equ. 6 cannot be solved directly, it has some nice properties. For instance, if a certain light ray has a large scaling factor k_i, which means it tends to have a large rendering error, the weight γ_i becomes large. The

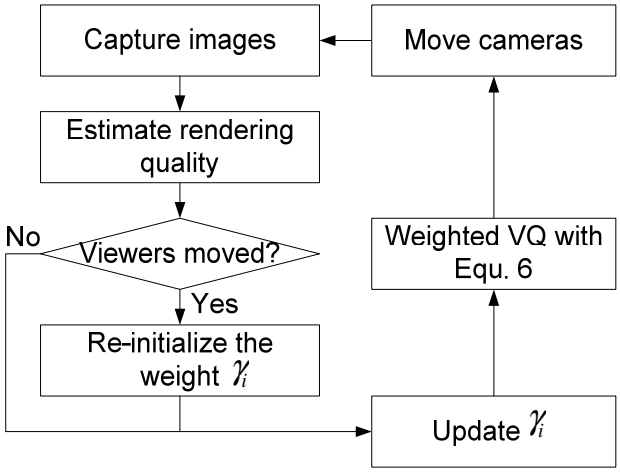

Fig. 3. The flow chart of our proposed active rearranged capturing algorithm for static or slowly-moving scenes.

vector quantization process will then reduce more on $\min_{j=1,\cdots,N}\|\mathbf{x}_i - \mathbf{c}_j\|^2$, effectively moving the capturing cameras closer to that light ray. Therefore, if the weights γ_i can be adjusted according to the rendering quality, the same weighted VQ algorithm can still be used to achieve the best rendering quality. These observations make it clear that we need an iterative solution for active rearranged capturing, which adjusts the weights γ_i according to some estimation of the final rendering quality.

C. A Recursive Algorithm for Active Rearranged Capturing

Fig. 3 shows the flow chart of the proposed recursive active rearranged capturing algorithm applicable for static or slowly-moving scenes. Given a set of newly captured images, we first estimate the rendering errors of all the light rays. If the viewers have moved (which can cause significant changes to the set of to-be-rendered light rays), the weights γ_i in Equ. 6 will be reinitialized. Afterwards, the weights γ_i are updated based on the estimated rendering quality. Weighted VQ is performed as Equ. 6, and the cameras are moved to capture the next set of images.

Two problems need to be addressed in the above flow chart, i.e., how to estimate the rendering quality given a set of captured images, and how to update the weights γ_i.

We propose to use the *local color consistency* as an estimation of the rendering quality. The local color consistency was first proposed in [19] for measuring the PIE functions. It states that a good rendering quality can be expected if the projected pixels used to interpolate a given light ray share the same intensity. Therefore, an easy implementation of the local color consistency is to use the variance of the projections. Take the light ray l_i in Fig. 2 as an example. The local color consistency $\mathcal{C}_i$ of l_i can be calculated as:

$$\mathcal{C}_i = \frac{1}{\sigma_i} \tag{7}$$

where σ_i is the variance of the projections of O to the K nearest neighboring images. Usually K is small, e.g., $K = 4$. In [19] it has been shown that such measurement is indeed a good estimate of the rendering quality. In the

follow discussions, we simply use $e_i \approx \sigma_i$ to model the rendering errors that we want to minimize in Equ. 6.

Now that the rendering quality has been estimated, we shall adjust the weights γ_i in Equ. 6 to find better locations for the capturing cameras. In the following, we present two weight updating mechanisms we experimented.

The first algorithm is based on the observation that if the weight associated with a certain light ray is increased, the weighted VQ algorithm that follows will tend to move the capturing cameras closer to the light ray. To improve the low quality light rays, we first define:

$$s_i = \log \sigma_i \tag{8}$$

as the score for each light ray. Let s_{min} and s_{max} be the minimum and maximum value of $s_i, i = 1, \cdots, L$. $\overline{s}$ be the average value of s_i. The weight γ_i^{t+1} at time instance $t+1$ is updated from those γ_i^t at time instance t as:

$$\gamma_i^{t+1} = \begin{cases} \gamma_i^t * (1 + (\xi - 1)\frac{\overline{s} - s_i}{\overline{s} - s_{min}}), & s_i \leq \overline{s}; \\ \gamma_i^t * (1 + (\zeta - 1)\frac{s_i - \overline{s}}{s_{max} - \overline{s}}), & s_i > \overline{s}. \end{cases} \tag{9}$$

where ξ and ζ are the minimum and maximum weight scaling factor. They are set as 0.5 and 4 respectively in the current implementation. As illustrated in Fig. 4, Equ. 9 says that if the variance of the projected pixels for a light ray is greater than the average (thus the local color consistency is bad), its weight will be increased. During the weighted VQ, the cameras will then move closer to that light ray. Otherwise, the cameras will move away. Note that after the weight update with Equ. 9, one should normalize the new weights such that $\sum_{i=1}^{L} \gamma_i^{t+1} = 1$.

Equ. 9 requires initial values of γ_i^0 to start the iteration. We find in practice the following initialization works well:

$$\gamma_i^0 \propto \min_{j=1,\cdots,N} \|\mathbf{x}_i - \mathbf{c}_j\|, \tag{10}$$

which gives higher weights to light rays that are far from any capturing cameras.

The second weight updating algorithm is based on the linear approximation between the rendering error and the minimum angular difference in Equ. 5, it is straightforward that:

$$\begin{aligned} \gamma_i &= w_i^2 k_i^2 \\ &\approx \frac{e_i^2}{\min_{j=1,\cdots,N} \|\mathbf{x}_i - \mathbf{c}_j\|^2} \\ &\approx \frac{\sigma_i^2}{\min_{j=1,\cdots,N} \|\mathbf{x}_i - \mathbf{c}_j\|^2} \end{aligned} \tag{11}$$

Since the rendering errors e_i are estimated from the local color consistency, the second weight updating algorithm does not require initial values of γ_i. That is, even if the viewers keep moving around, the weight reinitialization step in Fig. 3 can be skipped, making it more adaptable for viewer changes and dynamic scenes.

The recursive active rearranged capturing algorithm is thus summarized in Fig. 5.

D. Discussions

If we adopt the second weight updating algorithm in the previous section, the proposed recursive ARC algorithm is guaranteed to converge if all the assumptions made in the paper are correct. This can be seen as follows. The weight

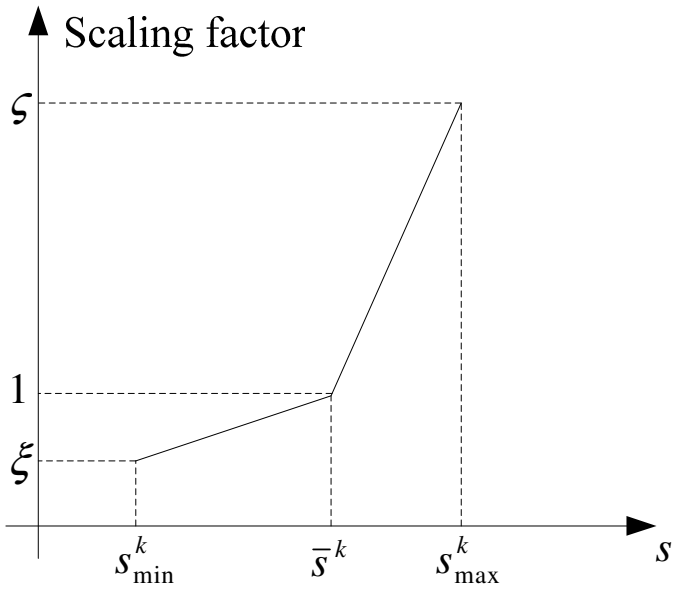

Fig. 4. Scaling factor for updating the auxiliary weights.

update step in Equ. 11 will not change the error function, while the weighted VQ that follows will guarantee to reduce the error function. Hence after each iteration the error function will be reduced, until it converges[1]. However, one may argue that although moving cameras closer to a rendered light ray may increase the rendering quality, the relationship may not be linear, hence Equ. 5 may not hold. Subsequently, after weighted VQ the rendering error may increase. In addition, although the local color consistency works well, it may not reflect the exact rendering error a light ray has.

In practice, we find both updating algorithm converges well for the sequences we tested. In fact, as shown in the next subsection, we always notice significant improvements after the first iteration. Further iterations only increase the rendering quality slightly. This characteristic surprises us but is good for active rearranged capturing, because it allows the algorithm to track viewer movements and dynamic scenes almost instantaneously, as long as the camera can move fast enough.

E. Synthetic experimental results

We verify the effectiveness of the proposed view-dependent active rearranged capturing algorithm with three synthetic scenes, namely, *teapot*, *vase* and *wineglass* (Fig. 7), all rendered from POV-Ray [28], which creates 3D photo-realistic images using ray tracing. The scenes contain complex textures, occlusions, semi-reflection, transparency, etc. In all cases, we use 64 cameras on the camera plane to capture the scene, as shown in Fig. 6. The initial camera locations are uniformly distributed on the camera plane. The virtual viewpoints can be anywhere

[1]Although weighed VQ guarantees convergence, it does not guarantee to converge to the global minimum. The same problem will exist in our algorithm.

for light field rendering, but for experimental purpose we place them on a plane in front of the camera plane, at a distance z_0, where z_0 roughly equals to the distance between neighboring initial camera locations d_0. The virtual viewpoints are assumed to be on a rectangular grid, indexed from 1 to 9, as shown in Fig. 6. The distance between neighboring virtual viewpoints is d.

We first consider the case where a single virtual view is rendered from the viewpoint indexed as 1 in Fig. 6. We use a constant depth plane as the geometric model. Fig. 8 shows the peak signal to noise ratio (PSNR) of the rendered image with respect to the number of iterations ARC performs. The improvement from no ARC (iteration number = 0) to ARC is very significant. Another interesting phenomenon is that the PSNRs improve dramatically when ARC was first applied (iteration number = 1), but then they improve very slowly when the number of iterations increases. This was unexpected but very useful for applying ARC to dynamic scenes and adapting to viewer movements.

Both weight updating algorithms work well and achieve similar performance. On the *teapot* scene, the second updating algorithm performs slightly better. This can be explained by Fig. 7, where we show some of the rendering results before and after ARC. The red dots in Fig. 7(c)(e)(g)$\cdots$ shows the projection of the camera locations to the virtual imaging plane of the rendering camera. It is interesting to observe that in *teapot*, many cameras are stuck in the pure black background when Equ. 9 is used to update the weights. In contrast, they immediately move to the foreground object when using Equ. 11. This is because although Equ. 9 adjust the weights up or down exponentially, it cannot directly set the weights of those background pixels to zero like Equ. 11 will do. Hence the convergence of ARC with Equ. 9 as the updating mechanism is slower. On the other hand, after 10 iterations, the first updating algorithm achieves slightly better PSNR on *vase* and *wineglass*. This is because the second algorithm relies on a linear approximation in Equ. 5, which is less flexible than the first algorithm in adjusting the weights.

We next examine the performance of ARC when multiple views are rendered. In Fig. 9, we show PSNR curves of *teapot* and *vase* with respect to the number of virtual viewpoints and their relative distances d (Fig. 6). In Fig. 9, if the number of viewpoints is P, we use all viewpoints whose indexes are less than or equal to P in Fig. 6. The weight updating algorithm in Equ. 11 is adopted in this comparison, with 3 iterations of ARC. Note in all cases, capturing with ARC produces significantly higher PSNR than if no ARC is performed. On the other hand, the improvement drops when the number of viewpoints increases. This is expected because when the number of viewpoints increases, ARC needs to seek balance between multiple views in order to achieve the best average rendering quality. Similarly, when the virtual views are more and more far apart, the to-be-rendered light rays have less overlap, which increases the difficulty for ARC to find camera locations that can render all the light rays well. Note the above conclusions are not definite (e.g. the *vase* scene), because different viewpoints may see completely different scene objects.

IV. A Self-Reconfigurable Camera Array

In this section, we present a self-reconfigurable camera array that makes use of the above developed ARC algorithm and adjusts the camera locations automatically based on the virtual views to be rendered. To the best of the authors' knowledge, this is the world's first camera array at such scale that can perform self-reconfiguration for

the purpose of image-based rendering.

A. System Overview

The system is composed of inexpensive off-the-shelf components. As shown in Fig. 10, there are 48 (8×6) Axis 205 network cameras placed on 6 linear guides. The linear guides are 1600 mm in length, thus the average distance between cameras is about 200 mm. Vertically the cameras are 150 mm apart. The cameras can capture up to 640×480 pixel images at maximally 30 fps. They have built-in HTTP servers, which send motion JPEG sequences upon request. The JPEG image quality is controllable. The cameras are connected to a central computer through 100Mbps Ethernet cables.

Each camera is mounted on a mobile platform (see Fig. 11), and is fastened to a pan servo capable of rotating through 90 degrees. They are mounted on a platform, which is equipped with another sidestep servo. The sidestep servo has been modified to rotate continuously. A gear wheel is attached to the sidestep servo, which allows the platform to move horizontally with respect to the linear guide. The gear rack is added to avoid slippery during the motion. The two servos on each camera unit allow the camera to have two degrees of freedom—pan and sidestep. However, the 12 cameras at the leftmost and rightmost columns have fixed positions and can only pan.

The servos are controlled by the Mini SSC II servo controller [29]. Each controller is in charge of up to 8 servos (either standard servos or modified ones). Multiple controllers can be chained, thus up to 255 servos can be controlled simultaneously through a single serial connection to a computer. In the current system, we use 11 Mini SSC II controllers to control 84 servos (48 pan servos, 36 sidestep servos).

Unlike any of the existing camera array systems mentioned in Section II-C, our whole system uses only a single computer. The computer is an Intel Xeon 2.4 GHz dual processor machine with 1GB of memory and a 32 MB NVIDIA Quadro2 EX graphics card. We developed a rendering algorithm that is very efficient and can perform region of interest identification, JPEG image decompression and camera lens distortion correction in real time (5-10 fps) [30].

Fig. 12(a) shows a set of images about a static scene captured by our camera array. The images are acquired at 320×240 resolution. The JPEG compression quality is set to be 30 (0 being the best quality and 100 being the worst quality). Each compressed image is about 12-18 Kbytes. In a 100 Mbps Ethernet connection, 48 cameras can send such JPEG image sequences to the computer simultaneously at 15-20 fps, which is satisfactory.

B. Software Architecture

The system software runs as two processes, one for capturing and the other for rendering. The capturing process is responsible for sending requests to and receiving data from the cameras. The received images (in JPEG compressed format) are directly copied to some shared memory that both processes can access. The capturing process is often lightly loaded, consuming about 20% of one processor in the computer. When the cameras start to move, their external calibration parameters need to be re-calculated in real-time. Camera calibration is also performed by the

capturing process. As will be described in the next section, calibration of the external parameters generally runs fast (150–180 fps).

The rendering process runs on the other processor. It is responsible for ROI identification, JPEG decoding, lens distortion correction, scene geometry reconstruction and novel view synthesis. Due to page limits, we will not discuss the real-time rendering aspect of the system. Interested readers are referred to [30] for more details.

C. Camera Calibration

Since our cameras are designed to be self-reconfigurable, calibration must be performed in real-time. Fortunately, the internal parameters of the cameras do not change during their motion, and can be calibrated offline. We use a large planar calibration pattern for the calibration process (Fig. 12). Bouguet's calibration toolbox [31] is used to obtain the internal camera parameters.

To calibrate the external parameters, we first need to extract the features on the checkerboard. We assume that the top two rows of feature points will never be blocked by the foreground objects. The checkerboard boundary is located by searching for the red strips of the board in the top region of the image. The pan servos are used here to make sure that the checkerboard is always centered in the captured images. Once the left and right boundaries are identified (as shown in Fig. 13), we locate the top two rows of corner features through linear filtering [30]. The feature positions are then refined to sub-pixel accuracy by finding the saddle points, as in [31]. The corners below the second row are extracted row by row. At each row, we predict the feature locations based on the previous two rows of features. The accurate positions of the features are then found through the same approach above. The results of such feature extraction are shown in Fig. 13. Note that if the corner detector cannot find a feature along a column for a certain row due to various reasons such as occlusions, it will stop finding features below that row in that column.

Finally, to obtain the 6 external parameters (3 for rotation and 3 for translation) of the cameras, we use the algorithm proposed by Zhang [32]. The Levenberg-Marquardt method implemented in MinPack [33] is adopted for the nonlinear optimization. The above calibration process runs very fast on our processor (150–180 fps at full speed). As long as there are not too many cameras moving around simultaneously, we can perform calibration on-the-fly during the camera movement. In the current implementation, we constrain movement to one camera per row at any time instance. After a camera has sidestepped, it will pan if necessary in order to keep the calibration board in the middle of the captured image.

D. Real-Time Rendering

We developed a real-time rendering algorithm that takes the 48 captured frames at any instance and renders virtual views at arbitrary viewpoints for dynamic scenes. Briefly speaking, the algorithm reconstructs a mesh model of the scene on the fly, and renders the virtual views with unstructured Lumigraph rendering [12]. The algorithm achieves 5-10 frames per second rendering speed on a moderate computer. The rendering software is downloadable from:

http://amp.ece.cmu.edu/projects/MobileCamArray/.

A thorough explanation of the rendering algorithm is out of the scope of this paper, and we refer the interested reader to [30].

E. Self-Reconfiguration of the Cameras

In Section III we presented an active rearranged capturing algorithm. There we assumed that all the capturing cameras can move freely on the camera plane. Such assumption is very difficult to implement in practical systems. In this section, we present a local rearrangement algorithm for the self-reconfiguration of the cameras, given that they are constrained on the linear guides.

Fig. 14 shows two real-world examples rendered from our camera array. Fig. 14(i-c) and (iii-c) shows the score as defined in Equ. 8 obtained during the rendering process. Note most errors are located around object boundaries and complex texture regions. The goal is again to move the cameras around to enhance the quality of the rendered light rays.

Our revised ARC algorithm contains the following steps:

1. *Locate the camera plane and the linear guides* (as line segments on the camera plane). The camera positions in the world coordinate are obtained through the calibration process. Although they are not strictly on the same plane, we use an approximated one which is parallel to the checkerboard. The linear guides are located by averaging the vertical positions of each row of cameras on the camera plane. As shown in Fig. 15, denote the vertical coordinates of the linear guides on the camera plane as $Y_j, j = 1, \cdots, 6$.

2. *Back-project the rendered light rays to the camera plane*. In Fig. 15, one such light ray was back-projected as (x_i, y_i) on the camera plane. Note the light rays can be sub-sampled to save computation.

3. *Collect score for each pair of neighboring cameras on the linear guides*. The capturing cameras on each linear guide naturally divide the guide into 7 segments. Let them be B_{jk}, where j is the row index of the linear guide, k is the index of bins on that guide, $1 \leq j \leq 6$, $1 \leq k \leq 7$. If a back-projected vertex (x_i, y_i) satisfies

$$Y_{j-1} < y_i < Y_{j+1} \quad and \quad x_i \in B_{jk}, \tag{12}$$

the score of the vertex is added to the bin B_{jk}. After all the vertices have been back-projected, we obtain a set of accumulated scores for each linear guide, denoted as S_{jk}, where j is the row index of the linear guide, k is the index of bins on that guide.

5. *Determine which camera to move on each linear guide*. Given a linear guide j, we look for the largest $S_{jk}, 1 \leq k \leq 7$. Let it be S_{jK}. If the two cameras forming the corresponding bin B_{jK} are not too close to each other, one of them will be moved towards the other (thus reducing their distance). Notice each camera is associated with two bins. To determine which one of the two cameras should move, we check their other associated bin and move the camera with a smaller accumulated score in its other associated bin.

6. *Move the cameras*. Once the moving cameras have been decided, we issue them commands such as "move

left" or "move right"[2]. Once the cameras are moved, the process waits until it is confirmed that the movement has finished and the cameras are re-calibrated. Then it jumps back to step 1 for the next epoch of movement.

The above algorithm can be viewed as a simplified version of the active rearranged capturing algorithm described in Section III. The three major simplifications are: 1. Cameras can only move on a line. Effectively the 2D weighted vector quantization algorithm becomes a 1d vector quantization problem. 2. Each time only one camera can move on a row. 3. The camera movement has fixed step size, since the only command we can issue is "move left" or "move right". The latter two simplifications may affect the convergence speed and performance of the algorithm. Nevertheless, such a simplified algorithm can still bring improvements on the rendering quality of the virtual views.

We show results of the proposed algorithm in Fig. 14. In Fig. 14 (i) and (iii), the capturing cameras are evenly spaced on the linear guide. Fig. 14(i) is rendered behind the camera plane, and Fig. 14(iii) is rendered in front of the camera plane. Due to depth discontinuities, some artifacts can be observed from the rendered images (Fig. 14 (i-d) and (iii-d)) along the object boundaries. Fig. 14(b) is the reconstructed depth of the scene at the virtual viewpoint. Fig. 14(c) is the rendering error score computed during the rendering process. It is obvious that along the object boundaries, the score is high, which indicates bad rendering quality. The red dots in Fig. 14(c) are the projections of the capturing camera positions to the virtual imaging plane.

Fig. 14 (ii) and (iv) shows the rendering result after ARC. Fig. 14 (ii) is the result of 6 epochs of camera movement, and Fig. 14 (iv) is after 20 epochs. It can be seen from the score map (Fig. 14(c)) that after the camera movement, the consistency gets better. The cameras have been moved, which is reflected as the red dots in Fig. 14(c). The cameras moves towards the regions where the score is high, which effectively increases the sampling rate around those regions. Fig. 14 (ii-d) and (iv-d) shows the rendering results after self-reconfiguration, which are much better than 14 (i-d) and (iii-d). A video clip on the ARC process is available at:

http://amp.ece.cmu.edu/projects/MobileCamArray/.

The major limitation of our self-reconfigurable camera array is that the motion of the cameras is slow. When the computer write a command to the serial port, the command will be buffered in the Mini SSC II controller for about 15 ms before sending to the servo. After the servo receives the command, there is also a long delay (hundreds of ms) before it moves enough distance. Therefore, during the self-reconfiguration of the cameras, we have to assume that the scene is either static or moving very slowly, and the viewer is not changing his/her viewpoint all the time. In our current implementation, the calibration process and the rendering process run separately. During self-reconfiguration, the moving cameras are not used for rendering until their calibration parameters are successfully computed. We observe some jittering artifacts of the rendered images when the moved cameras have not been fully calibrated.

There is no collision detection in the current system while moving the cameras. Although the calibration process is very stable and gives fairly good estimation of the camera positions, collision could still happen. In the previous

[2]We can only send such commands to the sidestep servos, because the servos were modified for continuous rotation. The positions of the cameras after movement is uncertain, and can only be obtained through the calibration process.

subsection, we have a threshold for verifying whether two cameras are too close to each other. The current threshold is set as 10 cm, which is reasonably safe in all our experiments.

V. Conclusion

This paper presented an active rearranged capturing algorithm for IBR. We made two major contributions in this paper. First, we showed that by observing the captured images at some initial positions, active rearranged capturing can refine the capturing positions using a recursive weighted vector quantization algorithm. After rearrangement, the rendering quality of the virtual views can be improved significantly. Second, we have presented the world's first large scale self-reconfigurable camera array, which consisted of 48 mobile cameras. Despite the constraints on how the cameras can move on the camera plane in practice, the rendering quality of our camera array was still shown to be better than the traditional static camera arrays.

Acknowledgment

The authors would like to thank Ted Square for proof-reading the paper. We also thank the anonymous reviewers for their valuable suggestions to improve the paper.

References

[1] C. Zhang and T. Chen, "A survey on image-based rendering - representation, sampling and compression," *EURASIP Signal Processing: Image Communication*, vol. 19, no. 1, pp. 1–28, 2004.

[2] H.-Y. Shum, S. Kang, and S.-C. Chan, "Survey of image-based representations and compression techniques," *IEEE Trans. CSVT*, vol. 13, no. 11, pp. 1020 – 1037, 2003.

[3] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong, "Plenoptic sampling," in *Proc. SIGGRAPH*, 2000, pp. 307–318.

[4] C. Zhang and T. Chen, "Spectral analysis for sampling image-based rendering data," *IEEE Trans. on CSVT*, vol. 13, no. 11, pp. 1038–1050, 2003.

[5] ——, "Generalized plenoptic sampling," *Carnegie Mellon Technical Report, AMP01-06*, 2001.

[6] ——, "Active scene capturing for image-based rendering," Carnegie Mellon University, Tech. Rep. AMP03-02, 2003.

[7] ——, "A system for active image-based rendering," in *Proc. ICME*, 2003.

[8] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. Horowitz, "The light field video camera," in *Proc. of Media Processors*, 2002.

[9] J. C. Yang, M. Everett, C. Buehler, and L. McMillan, "A real-time distributed light field camera," in *Eurographics Workshop on Rendering*, 2002.

[10] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. SIGGRAPH*, 1996, pp. 31–42.

[11] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," in *Proc. of SIGGRAPH*, 1999, pp. 299–306.

[12] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen, "Unstructured lumigraph rendering," in *Proc. SIGGRAPH*, 2001, pp. 425–432.

[13] Z. C. Lin and H. Y. Shum, "On the number of samples needed in light field rendering with constant-depth assumption," in *Proc. CVPR*, 2000.

[14] S. Fleishman, D. Cohen-Or, and D. Lischinski, "Automatic camera placement for image-based modeling," in *Computer Graphics Forum*, 2000.

[15] J. O'Rourke, *Art Gallery Theorems and Algorithms*, ser. The International Series of Monographs on Computer Science. Oxford University Press, 1987.

[16] T. Werner, V. Hlavác, A. Leonardis, and T. Pajdla, "Selection of reference views for image-based representation," in *Proc. ICPR*, 1996.

[17] R. Namboori, H. C. Teh, and Z. Huang, "An adaptive sampling method for layered depth image," in *Computer Graphics International (CGI)*, 2004.

[18] H. Schirmacher, W. Heidrich, and H. P. Seidel, "Adaptive acquisition of lumigraphs from synthetic scenes," in *Proc. EUROGRAPHICS*, 1999.

[19] C. Zhang and T. Chen, "Non-uniform sampling of image-based rendering data with the position-interval error (pie) function," in *Proc. VCIP*, 2003.

[20] T. Naemura, J. Tago, and H. Harashima, "Real-time video-based modeling and rendering of 3d scenes," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 66–73, 2002.

[21] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan, "Image-based visual hulls," in *Proc. SIGGRAPH*, 2000, pp. 369–374.

[22] G. G. Slabaugh, R. W. Schafer, and M. C. Hans, "Image-based photo hulls," HP Labs, Tech. Rep. HPL-2002-28, 2002.

[23] R. Yang, G. Welch, and G. Bishop, "Real-time consensus-based scene reconstruction using commodity graphics hardware," in *Proc. Pacific Graphics*, 2002.

[24] S. V. T. Kanade, H. Saito, "The 3d room: digitizing time-varying 3d events by synchronized multiple video streams," *Technical Report, CMU-RITR-98-34*, 1998.

[25] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High performance imaging using large camera arrays," in *Proc. SIGGRAPH*, 2005.

[26] S. Vedula, S. Baker, and T. Kanade, "Spatio-temporal view interpolation," in *13th ACM Eurographics Workshop on Rendering*, 2002.

[27] Y. Linde, A. Buzo, , and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communications*, vol. 28, no. 1, pp. 84–95, 1980.

[28] Pov-Ray, "http://www.povray.org."

[29] MiniSSC-II, "Scott edwards electronics inc., http://www.seetron.com/ssc.htm."

[30] C. Zhang, "On sampling of image-based rendering data," Ph.D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University, 2004.

[31] J.-Y. Bouguet, "Camera calibration toolbox for matlab, http://www.vision.caltech.edu/bouguetj/calib_doc/," 1999. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[32] Z. Zhang, "A flexible new technique for camera calibration," *Technical Report, MSR-TR-98-71*, 1998.

[33] J. J. Moré, "The levenberg-marquardt algorithm, implementation and theory," *G. A. Watson, editor, Numerical Analysis, Lecture Notes in Mathematics*, vol. 630, pp. 105–116, 1977.

Given a set of images newly captured, perform:

1) **Error estimation**, estimate the rendering error using local color consistency for any light ray l_i:

   Rendered color: $m_i = \sum_{k=1}^{K} \mu_k I_{ik}$
   Variance: $\sigma_i^2 = \sum_{k=1}^{K} \mu_k (I_{ik} - m_i)^2$

   where the summations are for the K nearest neighbors. μ_k is the weight for each captured light ray determined by the rendering algorithm, I_{ik} is the captured light ray intensity.

2) **Weight update**, update the weights γ_i as Equ. 9 or 11.

3) **Weighted VQ**, perform weighted vector quantization with a modified LBG-VQ algorithm [27]:

   a. **Nearest neighbor condition:**

   $$\mathbf{x}_i \in \mathcal{R}_j, \text{if } \|\mathbf{x}_i - \mathbf{c}_j\| \leq \|\mathbf{x}_i - \mathbf{c}_{j'}\|, \forall j' = 1, \cdots, N$$

   where $\mathcal{R}_j$ is the neighborhood region of centroid $\mathbf{c}_j$.

   b. **Centroid condition:**

   $$\mathbf{c}_j = \frac{\sum_{\mathbf{x}_i \in \mathcal{R}_j} \gamma_i \mathbf{x}_i}{\sum_{\mathbf{x}_i \in \mathcal{R}_j} \gamma_i}, j = 1, \cdots, N$$

4) **Capture images**, move the cameras according to the VQ result and capture images. Go back to 1).

Fig. 5. The recursive active rearranged capturing algorithm.

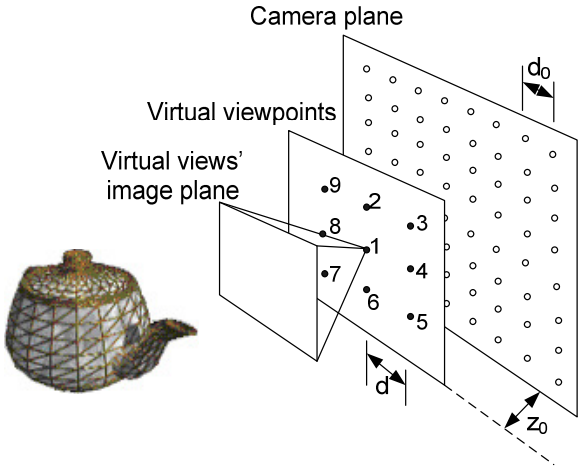

Fig. 6. Setup of experiments on synthetic scenes.

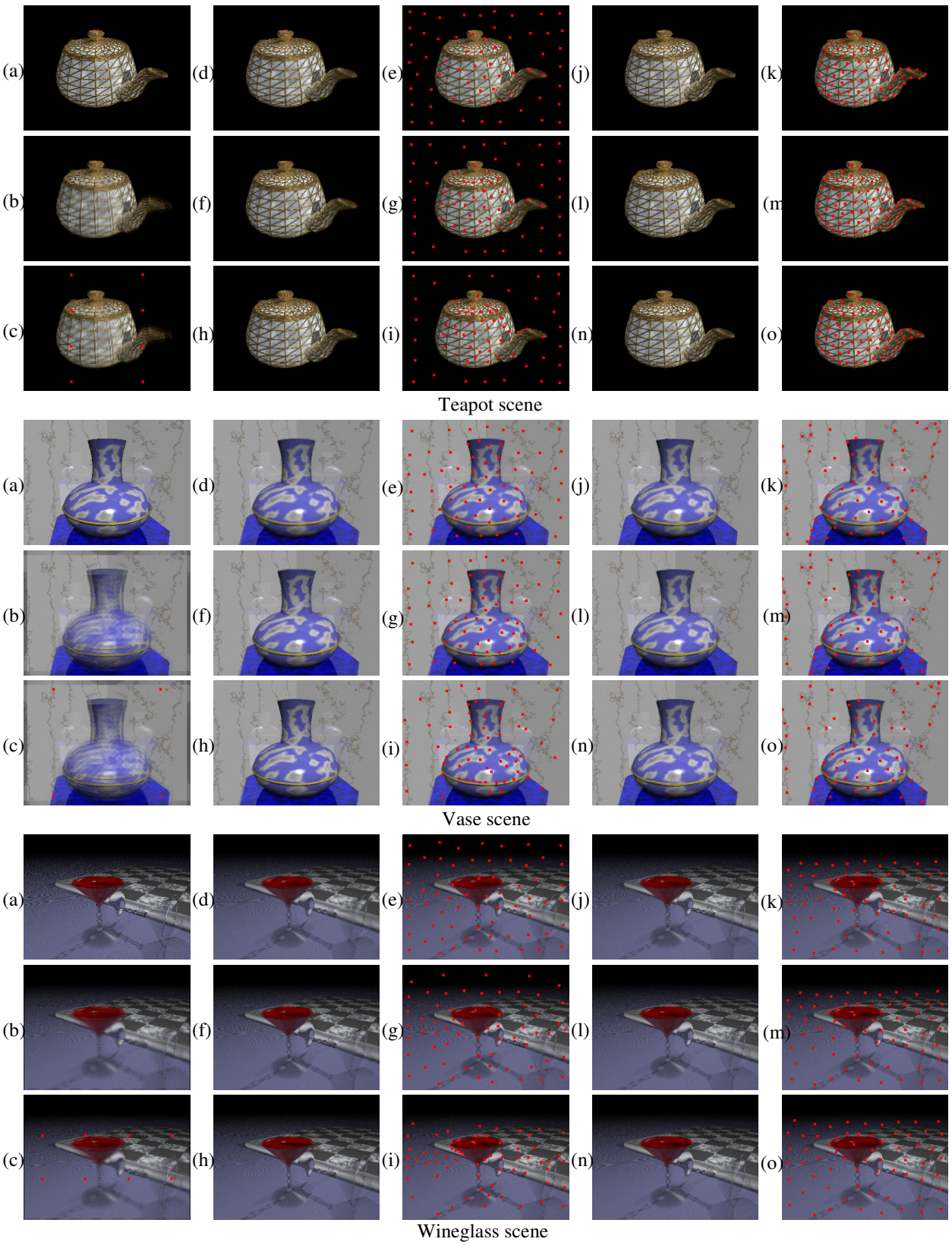

Fig. 7. Results of our active rearranged capturing (ARC) algorithm. (a) Virtual view to be rendered (ground truth). (b)(c) Rendering results before ARC and projection of the camera locations on the virtual imaging plane. (d)(e) Update with Equ. 9, 1 iteration of ARC. (f)(g) Update with Equ. 9, 3 iteration of ARC. (h)(i) Update with Equ. 9, 10 iteration of ARC. (j)(k) Update with Equ. 11, 1 iteration of ARC. (l)(m) Update with Equ. 11, 3 iteration of ARC. (n)(o) Update with Equ. 11, 10 iteration of ARC.

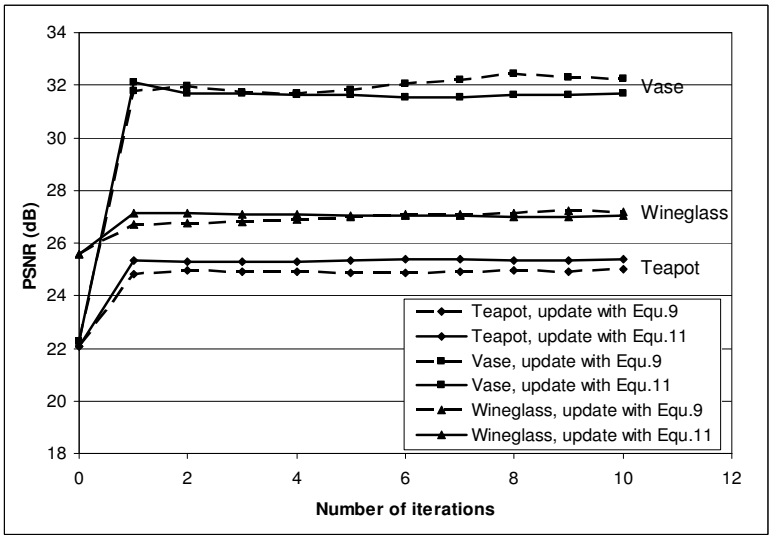

Fig. 8. PSNR of ARC with respect to the number of iterations. Each iteration contains weighted VQ, camera movement and new image set capturing.

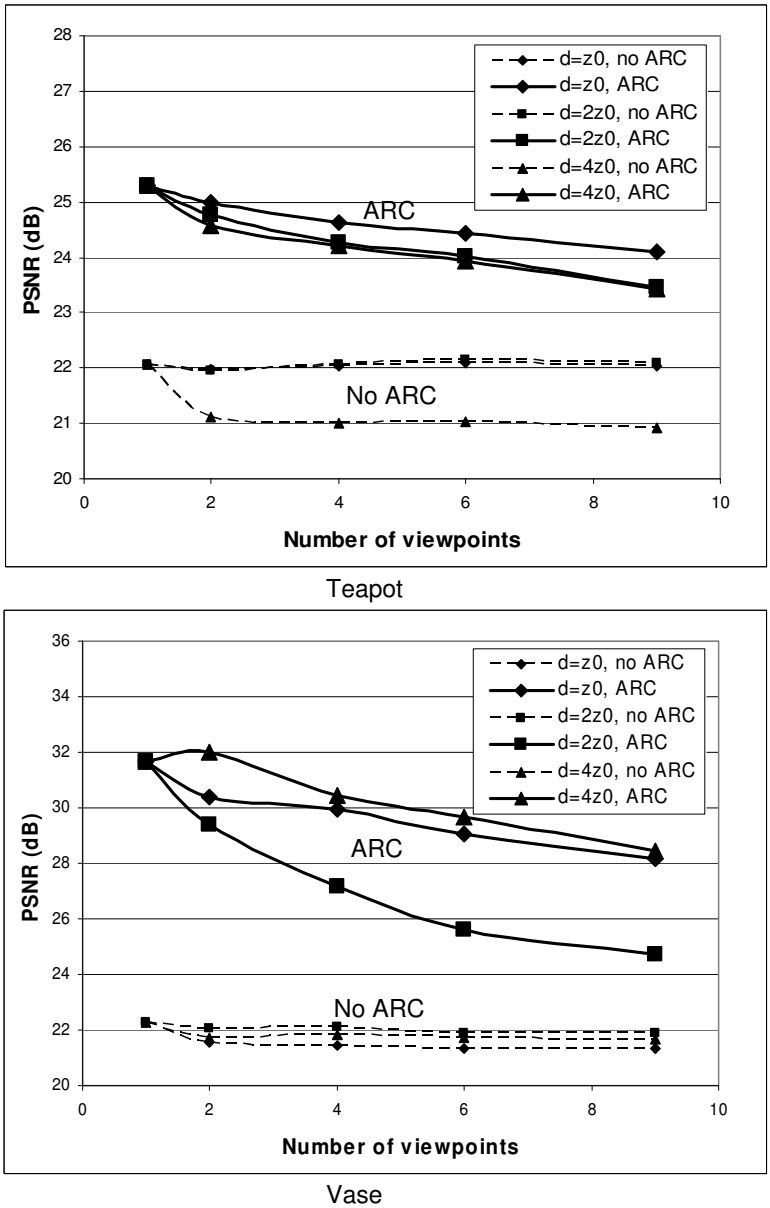

Fig. 9. Performance of ARC for multiple virtual viewpoints. The symbols such as d and z_0 were defined in Fig. 6.

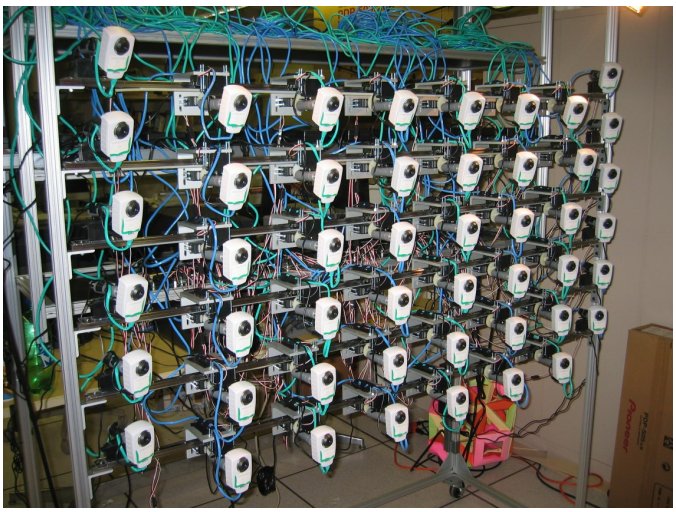

Fig. 10. Our self-reconfigurable camera array system with 48 cameras.

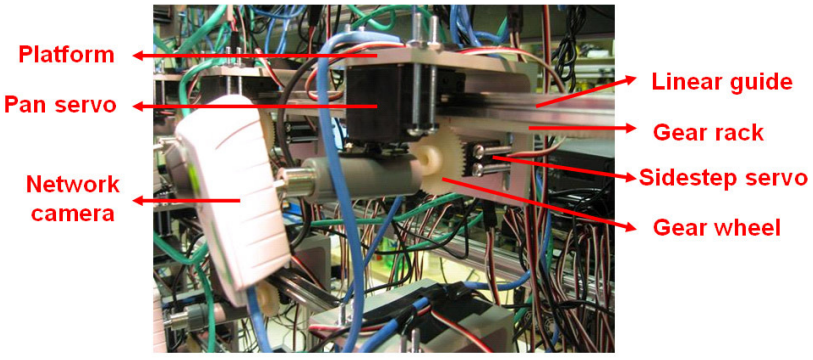

Fig. 11. The mobile camera unit.

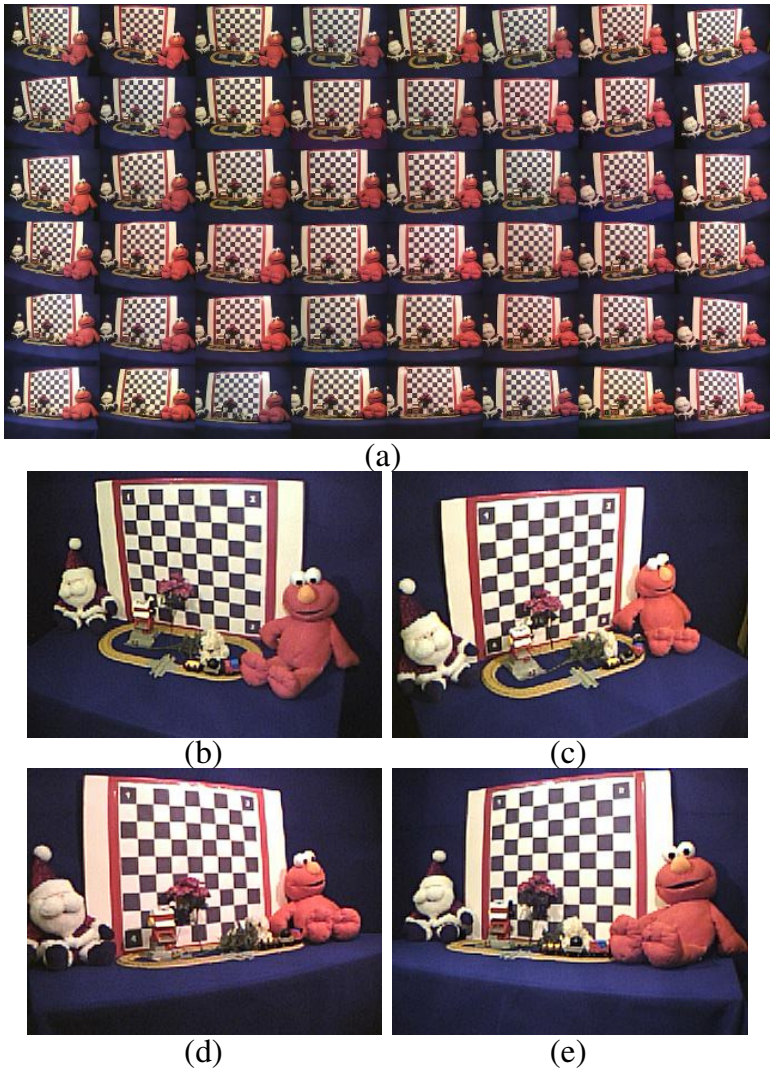

Fig. 12. Images captured by our camera array. (a) All the images; (b–e) sample images from selected cameras.

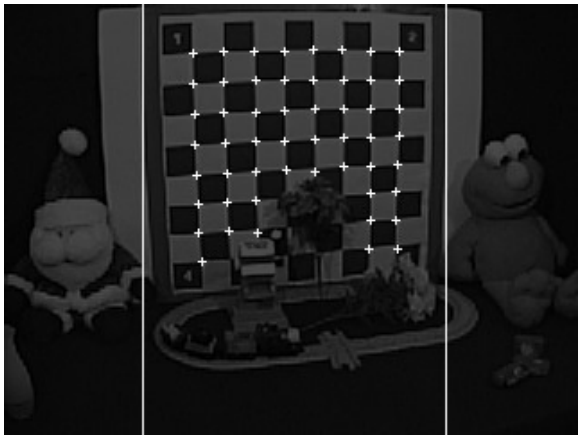

Fig. 13. Locate the feature corners of the calibration pattern.

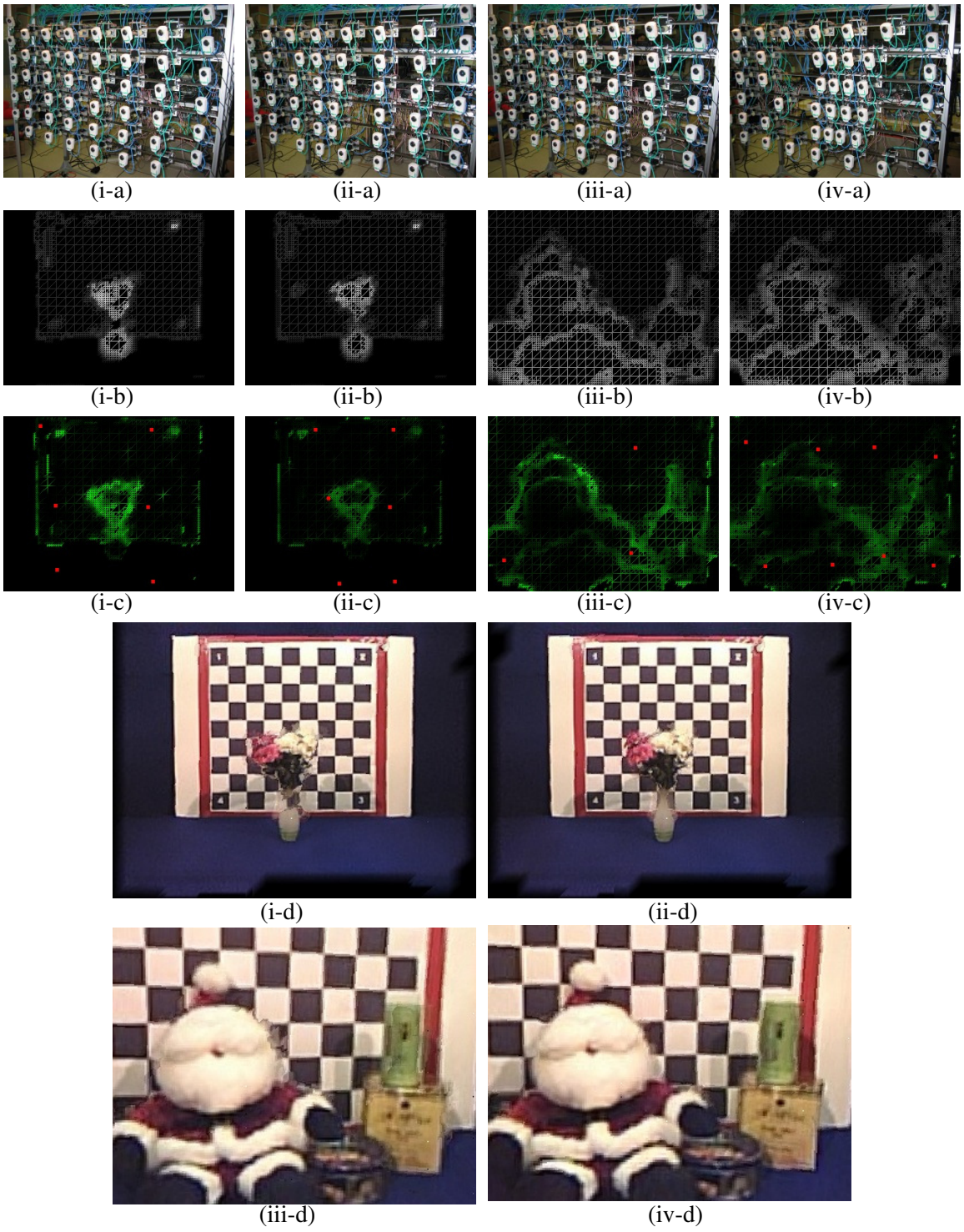

Fig. 14. Scenes rendered by reconfiguring our camera array. (i) Scene *flower*, cameras are evenly spaced; (ii) scene *flower*, cameras are self-reconfigured (6 epochs); (iii) scene *Santa*, cameras are evenly spaced; (iv) scene *Santa*, cameras are self-reconfigured (20 epochs); (a) the camera arrangement; (b) reconstructed depth map, brighter intensity means smaller depth; (c) the score of the mesh vertexes and the projection of the camera positions to the virtual imaging plane (red dots), darker intensity means better consistency; (d) rendered image.

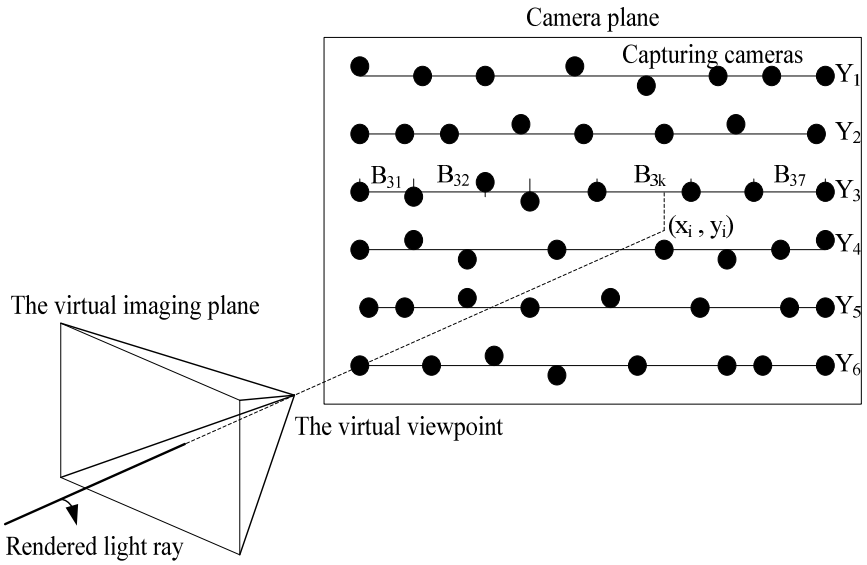

Fig. 15. Self-reconfiguration of the cameras.