Multi-View Imaging: Capturing and Rendering Interactive Environments

Cha Zhang
Microsoft Research
One Microsoft Way, Redmond WA 98052
Email: chazhang@microsoft.com

Tsuhan Chen
ECE, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh PA 15213
Email: tsuhan@cmu.edu

***Abstract*—This paper presents a self-reconfigurable camera array system that captures and renders 3D virtual scenes interactively. It is composed of an array of 48 cameras mounted on mobile platforms. We propose an efficient algorithm that is capable of rendering high-quality novel views from the captured images. The algorithm reconstructs a view-dependent multi-resolution 2D mesh model of the scene geometry on the fly and uses it for rendering. The algorithm combines region of interest (ROI) identification, JPEG image decompression, lens distortion correction, scene geometry reconstruction and novel view synthesis seamlessly on a single Intel Xeon 2.4 GHz processor, which is capable of generating novel views at 4–10 frames per second (fps). In addition, we present a view-dependent adaptive capturing scheme that moves the cameras in order to show even better rendering results. Such camera reconfiguration naturally leads to a nonuniform arrangement of the cameras on the camera plane, which is both view-dependent and scene-dependent.**

I. Introduction

Multi-view imaging, in particular, image-based rendering (IBR), has received a lot of interest recently [1], [2]. A number of camera arrays has been built for multi-view imaging in literature. For instance, Matusik et al. [3] used 4 cameras for rendering using image-based visual hull (IBVH). Yang et al. [4] had a 5-camera system for real-time rendering with the help of modern graphics hardware; Schirmacher et al. [5] built a 6-camera system for on-the-fly processing of generalized Lumigraphs; Naemura et al. [6] constructed a system of 16 cameras for real-time rendering. Several large arrays consisting of tens of cameras have also been built, such as the Stanford multi-camera array [7], the MIT distributed light field camera [8] and the CMU 3D room [9]. These three systems have 128, 64 and 49 cameras, respectively.

In the above camera arrays, those with a small number of cameras can usually achieve real-time rendering [3], [4]. On-the-fly geometry reconstruction is widely adopted to compensate for the lack of cameras, and the viewpoint is often limited. Large camera arrays, despite their increased viewpoint ranges, often have difficulty in achieving satisfactory rendering speed due to the large amount of data to be handled. The Stanford system focused on grabbing synchronized video sequences onto hard drives. It certainly can be used for real-time rendering but no such results have been reported in literature. The CMU 3D room was able to generate good-quality novel views both spatially and temporarily [10]. 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 unavoidable according to plenoptic sampling analysis [11], [12].

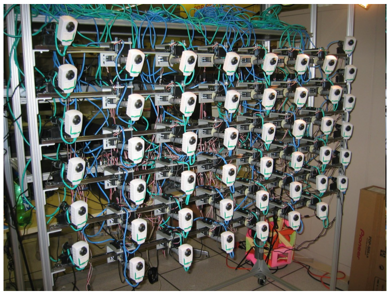


Fig. 1. Our self-reconfigurable camera array system with 48 cameras.

In this paper, we present a large self-reconfigurable camera array consisting of 48 cameras, as shown in Figure 1. We first propose an efficient rendering algorithm that generates high-quality virtual views by reconstructing the scene geometry on-the-fly. Differing from previous work [4], [5], the geometric representation we adopted is a view-dependent multi-resolution 2D mesh with depth information on its vertices. This representation greatly reduces the computational cost of geometry reconstruction, making it possible to be performed on-the-fly during rendering.

Compared with existing camera arrays, our system has a unique characteristic—the cameras are reconfigurable. They can both sidestep and pan during the capturing and rendering process. This capability makes it possible to reconfigure the arrangement of the cameras in order to achieve better rendering results. This paper also presents an algorithm that automatically moves the cameras based on the rendering quality of the synthesized virtual view. Such camera reconfiguration leads to a nonuniform arrangement of the cameras on the camera plane, which is both view-dependent and scene-dependent.

The paper is organized as follows. Related work is reviewed in Section II. Section III presents an overview of our camera array system. The calibration issue is discussed in Section IV. The real-time rendering algorithm is presented in detail in Section V. The self-reconfiguration of the camera positions is discussed in Section VI. We present our conclusions in Section VII.

II. Related Work

In IBR, when the number of captured images for a scene is limited, adding geometric information can significantly improve the rendering quality. In fact, there is a geometry-image continuum which covers a wide range of IBR techniques, as is surveyed in [1]. In practice, an accurate geometric model is often difficult to attain, because it requires much human labor. Many approaches in literature assume a known geometry, or acquire the geometry via manual assistance or a 3D scanner. Recently, there has been increasing interest in on-the-fly geometry reconstruction for IBR [5], [3], [4] .

Depth from stereo is an attractive candidate for geometry reconstruction in real-time. Schirmacher et al. [5] built a 6-camera system which was composed of 3 stereo pairs and claimed that the depth could be recovered on-the-fly. However, each stereo pair needed a dedicated computer for the depth reconstruction, which is expensive to scale when the number of input cameras increases. Naemura et al. [6] 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. Such a depth map is computed with respect to a fixed viewpoint; thus the synthesized view is sensitive to geometry reconstruction errors. Another constraint of stereo based algorithms is that the input images need to be pair-wise positioned or rectified, which is not convenient in practice.

Matusik et al. [3] proposed image-based visual hull (IBVH), which rendered dynamic scenes in real-time from 4 cameras. IBVH is a clever algorithm which computes and shades the visual hull of the scene without having an explicit visual hull model. The computational cost is low thanks to an efficient pixel traversing scheme, which can be implemented with software only. Another similar work is the polyhedral visual hull [13], which computes an exact polyhedral representation of the visual hull directly from the silhouettes. Lok [14] and Li et al. [15] reconstructed the visual hull on modern graphics hardware with volumetric and image-based representations. One common issue of visual hull based rendering algorithms is that they cannot handle concave objects, which makes some close-up views of concave objects unsatisfactory.

An improvement over the IBVH approach is the image-based photo hull (IBPH) [16]. IBPH utilizes the color information of the images to identify scene geometry, which results in more accurately reconstructed geometry. Visibility was considered in IBPH by intersecting the visual hull geometry with the projected line segment of the considered light ray in a view. Similar to IBVH, IBPH requires the scene objects' silhouettes to provide the initial geometric information; thus, it is not applicable to general scenes (where extracting the silhouettes could be difficult) or mobile cameras.

Recently, Yang et al. [4] proposed a real-time consensus-based scene reconstruction method using commodity graphics hardware. Their algorithm utilized the Register Combiner for color consistency verification (CCV) with a sum-of-square-difference (SSD) measure, and obtained a per-pixel depth map in real-time. Both concave and convex objects of general scenes could be rendered with their algorithm. However, their recovered depth map could be very noisy due to the absence of a convolution filter in commodity graphics hardware.

As modern computer graphics hardware becomes more and more programmable and powerful, the migration to hardware geometry reconstruction (HGR) algorithms is foreseeable. However, at the current stage, HGR still has many limitations. For example, the hardware specification may limit the maximum number of input images during the rendering [15], [4]. Algorithms that can be used on hardware are constrained. For instance, it is not easy to change the CCV in [4] from SSD to some more robust ones such as pixel correlations. When the input images have severe lens distortions, the distortions must be corrected using dedicated computers before the images are sent to the graphics hardware.

Self-reconfiguration of the cameras is a form of non-uniform sampling (or adaptive capturing) of IBR scenes. In [17], Zhang and Chen proposed a general non-uniform sampling framework called the *Position-Interval-Error* (PIE) function. The PIE function led to two practical algorithms for capturing IBR scenes: progressive capturing (PCAP) and rearranged capturing (RCAP). PCAP captures the scene by progressively adding cameras at the places where the PIE values are maximal. RCAP, on the other hand, assumes that the overall number of cameras is fixed and tries to rearrange the cameras such that rendering quality estimated through the PIE function is the worst. A small scale system was developed in [18] to demonstrate the PCAP approach. The work by Schirmacher et al. [19] shared similar ideas with PCAP, but they only showed results on synthetic scenes.

One limitation about the above mentioned work is that the adaptive capturing process tries to minimize the rendering error everywhere as a whole. Therefore for a specific virtual viewpoint, the above work does not guarantee better rendering quality. Furthermore, since different viewpoints may require different camera configurations to achieve the best rendering quality, the final arrangement of the cameras is a tradeoff of all the possible virtual viewpoints, and the improvement over uniform sampling was not easy to show.

We recently proposed the view-dependent non-uniform sampling of IBR scenes [20]. Given a set of virtual views, the positions of the capturing cameras are rearranged in order to obtain the optimal rendering quality. The problem is formulated as a recursive weighted vector quantization problem, which can be solved efficiently. In that work we assume that all the capturing cameras can move freely on the camera plane. Such assumption is very difficult to implement in practical systems. This paper proposes a new algorithm for the self-reconfiguration of the cameras, given that they are constrained on the linear guides.

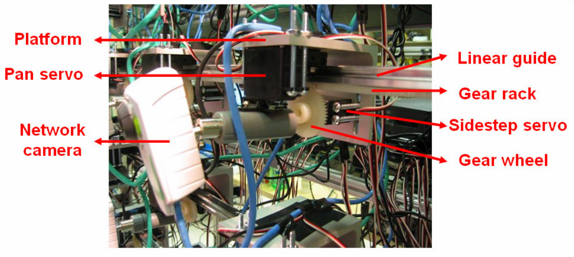


Fig. 2. The mobile camera unit.

III. Overview of the Camera Array System

A. Hardware

Our camera array system (as shown in Figure 1) is composed of inexpensive off-the-shelf components. 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. They can capture at a rate of up to 640 × 480 × 30fps. The cameras have built-in HTTP servers, which respond to HTTP requests and send out motion JPEG sequences. The JPEG image quality is controllable. The cameras are connected to a central computer through 100Mbps Ethernet cables.

The cameras are mounted on a mobile platform, as shown in Figure 2. Each camera is attached to a pan servo, which is a standard servo capable of rotating 90 degrees. They are mounted on a platform, which is equipped with another sidestep servo. The sidestep servo is hacked so that it can 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 slipping. 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 [21]. Each controller is in charge of no more than 8 servos (either standard servos or hacked 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 altogether 11 Mini SSC II controllers to control 84 servos (48 pan servos, 36 sidestep servos).

Unlike any of the existing camera array systems described in Section I, our system uses only one 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. As will be detailed in Section V, our rendering algorithm is so efficient that the ROI identification, JPEG image decompression and camera lens distortion correction, which were usually performed with dedicated computers in previous systems, can all be conducted during the rendering process for a camera array at our scale. On the other hand, it is not difficult to modify our system and attribute ROI identification and image decoding to dedicated computers, as was done in the MIT distributed light field camera [8].

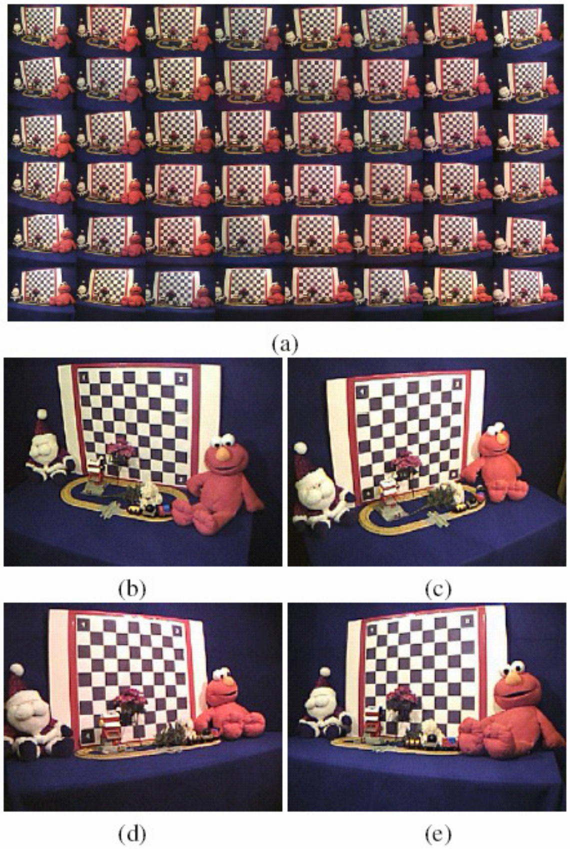


Fig. 3. Images captured by our camera array. (a) All the images. (b)(c)(d)(e) Sample images from selected cameras.

Figure 3 (a) shows a set of images for a static scene captured by our camera array. The images are acquired at 320×240 pixel. The JPEG compression quality factor is set to be 30 (0 being the best quality and 100 being the worst quality, according to the Axis camera's specification). 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. Several problems can be spotted from these images. First, the cameras have severe lens distortions, which has to be corrected during the rendering. Second, the colors of the captured images have large variations. The Axis 205 camera does not have flexible lighting control settings. We use the ”fixed indoor” white balance and ”automatic” exposure control in our system. Third, the disparity between cameras is large. As will be shown later, using a constant depth assumption to render the scene will generate images with severe ghosting artifacts. Finally, the captured images are noisy (Figure 3 (b)–(e)). This noise comes from both the CCD sensors of the cameras and the JPEG image compression. This noise brings an additional challenge to the scene geometry reconstruction.

The Axis 205 cameras cannot be easily synchronized. We

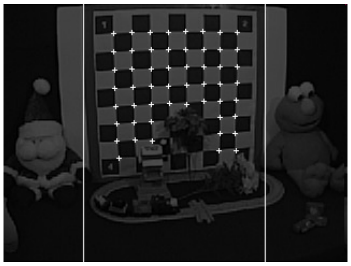


Fig. 4. Locate the features of the calibration pattern.

make sure that the rendering process will always use the most recently arrived images at the computer for synthesis. Currently we ignore the synchronization problem during the geometry reconstruction and rendering, though it does cause problems when rendering fast moving objects, as might have been observed in the submitted companion video files.

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 very lightweight, consuming about 20% of the CPU time of one of the processors in the computer. When the cameras start to move, their external calibration parameters need to be 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. Details about the rendering process will be described in Section V.

IV. Camera calibration

Since our cameras are designed to be mobile, 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 (Figure 3). Bouguet's calibration toolbox [22] is used to obtain the internal camera parameters.

To calibrate the external parameters, we first extract the feature positions on the checkerboard using two simple linear filters. The positions are then refined to sub-pixel accuracy by finding the saddle points, as in [22]. The results of feature extraction is shown in Figure 4. Notice that due to occlusions, not all the corners on the checkerboard can be extracted. However, calibration can still be performed using the extracted corners.

To obtain the 6 external parameters (3 for rotation and 3 for translation) of the cameras, we use the algorithm proposed by Zhang [23]. The Levenberg-Marquardt method implemented in MinPack [24] is used 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 impose the constraint that at any instance at most one camera on each row can sidestep. After a camera has sidestepped, it will pan if necessary in order to keep the calibration board in the middle of the captured image.

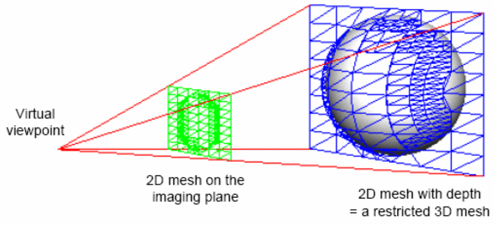

Virtual viewpoint
2D mesh on the imaging plane
2D mesh with depth = a restricted 3D mesh

Fig. 5. The multi-resolution 2D mesh with depth information on its vertices.

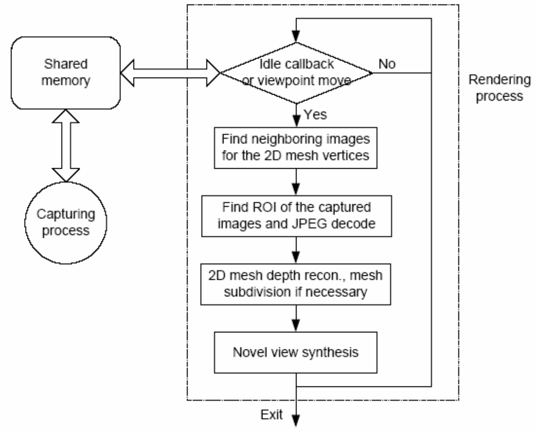

Shared memory
Capturing process
Idle callback or viewpoint move
No
Yes
Find neighboring images for the 2D mesh vertices
Find ROI of the captured images and JPEG decode
2D mesh depth recon., mesh subdivision if necessary
Novel view synthesis
Rendering process
Exit

Fig. 6. The flow chart of the rendering algorithm.

V. Real Time Rendering

A. *Flow of the rendering algorithm*

In this paper, we propose to reconstruct the geometry of the scene as a 2D multi-resolution mesh (MRM) with depths on its vertices, as shown in Figure 5. The 2D mesh is positioned on the imaging plane of the virtual view; thus, the geometry is view-dependent (similar to that in [4], [16], [3]). The MRM solution significantly reduces the amount of computation spent on depth reconstruction, making it possible to be implemented efficiently in software.

The flow chart of the rendering algorithm is shown in Figure 6. A novel view is rendered when there is an idle callback or the user moves the viewpoint. We first construct an initial sparse and regular 2D mesh on the imaging plane of the virtual view, as shown in Figure 7. This sparse mesh is used to obtain an initial estimate of the scene geometry. For each vertex of the 2D mesh, we first look for a subset of images that will be used to interpolate its intensity during the rendering. This

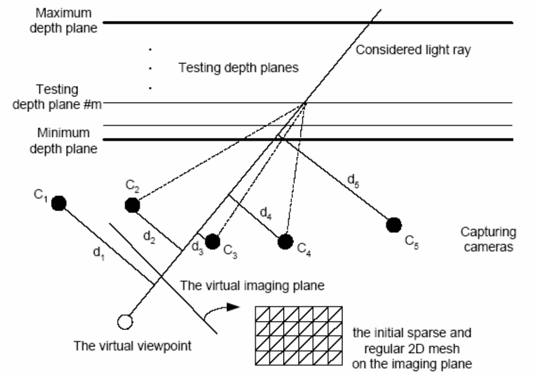


Fig. 7. Locate the neighboring images for interpolation and depth reconstruction through plane sweeping.

step has two purposes. First, we may use such information to identify the ROIs of the captured images and decode them when necessary, as is done in the next step. Second, only the neighboring images will be used for color consistency verification during the depth reconstruction, which is termed *local color consistency verification* (detailed in Section V-D). We then obtain the depths of the vertices in the initial 2D mesh through a plane-sweeping algorithm. At this stage, the 2D mesh can be used for rendering already; however, it may not have enough resolution along the object boundaries. We next perform a subdivision of the mesh in order to avoid the resolution problem at object boundaries. If a certain triangle in the mesh bears large depth variation, which implies a possible depth error or object boundary, subdivision is performed to obtain more detailed depth information. Afterwards, the novel view can be synthesized through multi-texture blending, similar to the unstructured Lumigraph rendering (ULR) [25]. Lens distortion is corrected in the last stage, although we also compensate the distortion during the depth reconstruction stage. Details of the proposed algorithm will be presented next.

B. Finding close-by images for the mesh vertices

Each vertex on the 2D mesh corresponds to a light ray that starts from the virtual viewpoint and passes through the vertex on the imaging plane. During the rendering, it will be interpolated from several light rays from nearby captured images. We need to identify these nearby images for selective JPEG decoding and the scene geometry reconstruction. Unlike the ULR [25] and the MIT distributed light field camera [8] where the scene depth is known, we do not have such information at this stage, and cannot locate the neighboring images by angular differences of the light rays[1]. Instead, we adopted the distance from the cameras' center of projection to the considered light ray as the criterion. As shown in Figure 7, the capturing cameras C_2, C_3 and C_4 have the smallest distances, and will be selected as the 3 closest images. As our cameras are roughly arranged on a plane and all point in roughly the same direction, when the scene is reasonably far from the capturing cameras, this distance measure is a good approximation of the angular difference used in the literature, yet it does not require the scene depth information.

C. ROI Identification and JPEG decoding

On the initial coarsely-spaced regular 2D mesh, if a triangle has a vertex that selects input image #*n* from one of the nearby cameras, the rendering of that triangle will need image #*n*. In other words, once all the vertices have found their nearby images, we will know which triangles require which images. This information is used to identify the ROIs of the images that need to be decoded.

We back-project the triangles that need image #*n* for rendering from the virtual imaging plane to the minimum depth plane and the maximum depth plane, and then project the resulting regions to image #*n*. The ROI of image #*n* is the smallest rectangular region that includes both of the projected regions. Afterwards, the input images that do not have an empty ROI will be JPEG decoded (partially).

D. Scene depth reconstruction

We reconstruct the scene depth of the light rays passing through the vertices of the 2D mesh using a plane sweeping method. Similar methods have been used in a number of previous algorithms [26], [27], [8], although they all reconstruct a dense depth map of the scene. As illustrated in Figure 7, we divide the world space into multiple testing depth planes. For each light ray, we assume the scene is on a certain depth plane, and project the scene to the nearby input images obtained in Section 3.3. If the assumed depth is correct, we expect to see consistent colors among the projections. The plane sweeping method sweeps through all the testing depth planes, and obtains the scene depth as the one that gives the highest color consistency.

There is an important difference between our method and previous plane sweeping schemes [26], [27], [8]. In our method, the CCV is carried out only among the nearby input images, not all the input images. We term this *local color consistency* verification. As the light ray is interpolated from only the nearby images, local CCV is a natural approach. In addition, it has some benefits over the traditional one. First, it is fast because we perform many fewer projections for each light ray. Second, it enables us to reconstruct geometry for non-diffuse scenes to some extent, because within a certain neighborhood, color consistency may still be valid even in non-diffuse scenes. Third, when CCV is performed only locally, problems caused by object occlusions during geometry reconstruction become less severe.

Care must be taken in applying the above method. First, the location of the depth planes should be equally spaced in the disparity space instead of in depth. This is a direct result from the sampling theory by Chai et al. [11]. In the same paper they also develop a sampling theory on the relationship between the number of depth planes and the number of captured images, which is helpful in selecting the number of depth planes. Second, when projecting the test depth planes to the

[1] Although it is possible to find the neighboring images of the light rays for each hypothesis depth plane, we found such an approach too time-consuming.

neighboring images, lens distortion must be corrected. Third, to improve the robustness of the color consistency matching among the noisy input images, a patch on each nearby image is taken for comparison. The patch window size relies heavily on the noise level in the input images. In our current system, the input images are very noisy. We have to use a large patch window to compensate for the noise. The patch is first down-sampled horizontally and vertically by a factor of 2 to reduce some of the computational burden. Different patches in different input images are then compared to generate an overall CCV score. Fourth, as our cameras have large color variations, color consistency measures such as SSD do not perform very well. We applied mean-removed correlation coefficient for the CCV. The correlation coefficients for all pairs of nearby input images are first obtained. The overall CCV score of the nearby input images is one minus the average correlation coefficient of all the image pairs. The depth plane resulting in the lowest CCV score is then selected as the scene depth.

The depth recovery process starts with an initial regular and sparse 2D mesh, as was shown in Figure 7. The depths of its vertices are obtained with the mentioned described above. The sparse mesh with depth can serve well during rendering if the depth of the scene does not vary much. However, if the scene depth does change, a dense depth map is needed around those regions for satisfactory rendering results. We subdivide a triangle in the initial mesh if its three vertices have large depth variation. For example, let the depths of a triangle's three vertices be d_{m_1}, d_{m_2} and d_{m_3}, where m_1, m_2, m_3 are the indices of the depth planes. We subdivide this triangle if:

$$\max_{p,q\in\{1,2,3\},p\neq q} |m_p - m_q| > T \qquad (1)$$

where T is a threshold set equal to 1 in the current implementation. During the subdivision, the midpoint of each edge of the triangle is selected as a new vertice, and the triangle is subdivided into 4 smaller ones. The depths of the new vertices are reconstructed under the constraints that they have to use the neighboring images of the three original vertices, and their depth search range is limited to the minimum and maximum depth of the original vertices. Other than Equation 1, the subdivision may also stop if the subdivision level reaches a certain preset limit.

Real-time, adaptive conversion from dense depth map or height field to a mesh representation has been studied in literature [28]. However, these algorithms assumed that a dense depth map or height field was available before hand. In contrast, our algorithm computes a multi-resolution mesh model directly during the rendering. The size of each triangles in the initial regular 2D mesh cannot be too large, since otherwise we may miss certain depth variations in the scene. A rule of thumb is that the size of the initial triangles/grids should match that of the object features in the scene. In the current system, the initial grid size is about 1/25 of the width of the input images. Triangle subdivision is limited to no more 2 levels.

E. Novel view synthesis

After the multi-resolution 2D mesh with depth information on its vertices has been obtained, novel view synthesis is easy. Our rendering algorithm is very similar to the one in ULR [25], except that our imaging plane has already been triangulated. Only the ROIs of the input images will be used to update the texture memory when a novel view is rendered. As the input images of our system have severe lens distortions, we cannot use the 3D coordinates of the mesh vertices and the texture matrix in graphics hardware to specify the texture coordinates. Instead, we perform the projection with lens distortion correction ourselves and provide 2D texture coordinates to the rendering pipeline. Fortunately, such projections to the nearby images have already been calculated during the depth reconstruction stage and can simply be reused.

F. Rendering results

We have used our camera array system to capture a variety of scenes, both static and dynamic. The speed of rendering process is about 4-10 fps, depending on many factors such as the number of testing depth planes used for plane sweeping, the patch window size for CCV, the initial coarse regular 2D mesh grid size, the number of subdivision levels used during geometry reconstruction and the scene content. For the scenes we have tested, the above parameters can be set to fixed values. For instance, our default setting is 12 testing depth planes for depth sweeping, 15×15 patch window size, 1/25 of the width of the input images as initial grid size, and maximally 2 level of subdivision.

The time spent on each step of the rendering process under the above default settings is as follows. Finding neighboring images and their ROI's takes less than 10 ms. JPEG decoding takes 15-40 ms. Geometry reconstruction takes about 80-120 ms. New view synthesis takes about 20 ms.

The rendering results of some static scenes are shown in Figure 9. In these results the cameras are evenly spaced on the linear guide. Figure 9(a)(b)(c) are results rendered with the constant depth assumption. The ghosting artifacts are very severe, because the spacing between our cameras is larger than most previous systems [8], [6]. Figure 9(d) is the result from the proposed algorithm. The improvement is significant. Figure 9(e) shows the reconstructed 2D mesh with depth information on its vertices. The grayscale intensity represents the depth – the brighter the intensity, the closer the vertex. Like many other geometry reconstruction algorithms, the geometry we obtained contains some errors. For example, in the background region of the *toys* scene, the depth should be flat and far, but our results have many small "bumps". This is because part of the background region has no texture, and thus is prone to error for depth recovery. However, the rendered results are not affected by these errors because we use view-dependent geometry and the local color consistency always holds at the viewpoint.

The performance of our camera array system on dynamic scenes is demonstrated in the companion video sequences. In general the scenes are rendered at high quality. The user is free to move the viewpoint and the view-direction when the scene object is also moving, which brings very rich new experiences.

G. Discussions

Our current system has certain hardware limitations. For example, the images captured by the cameras are at 320×240 pixel and the image quality is not very high. This is mainly constrained by the throughput of the Ethernet cable. Upgrading the system to Gigabit Ethernet or using more computers to handle the data could solve this problem. For dynamic scenes, we notice that our system cannot catch up with very fast moving objects. This is due to the fact that the cameras are not synchronized.

We find that when the virtual viewpoint moves out of the range of the input cameras, the rendering quality degrades quickly. A similar effect was reported in [8], [29]. The poor extrapolation results are due to the lack of scene information in the input images during the geometry reconstruction.

Since our geometry reconstruction algorithm resembles the traditional window-based stereo algorithms, it shares some of the same limitations. For instance, when the scene has a large depth discontinuity, our algorithm does not perform very well along the object boundary (especially when both foreground and background objects have strong textures). In the current implementation, our correlation window is very large in order to handle the noisy input images. Such a big correlation window tends to smooth the depth map. Figure 10 (i-d) and (iii-d) shows the rendering results of two scenes with large depth discontinuities. Notice the artifacts around the boundaries of the objects. To solve this problem, one may borrow ideas from the stereo literature [30], [31], which will be our future work. Alternatively, since we have built a mobile camera array, we may reconfigure the arrangement of the cameras, as will be described in the next section.

VI. Self-Reconfiguration of the Camera Positions

A. The proposed algorithm

Figure 10 (i-c) and (iii-c) shows the CCV score obtained while reconstructing the scene depth (Section V-D). It is obvious that if the consistency is bad (high score), the reconstructed depth tends to be wrong, and the rendered scene tends to have low quality. Our camera self-reconfiguration (CSR) algorithm thus tries to move the cameras to places where the CCV score is high.

Our CSR 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 world coordinates 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 Figure 8, we denote the vertical coordinates of the linear guides on the camera plane as $Y_j, j = 1, \cdots, 6$.

2. *Back-project the vertices of the mesh model to the camera plane*. Although during depth reconstruction the mesh can be subdivided, during this process we only make use of the initial sparse mesh (Figure 7). In Figure 8, one mesh vertex was back-projected as (x_i, y_i) on the camera plane. Notice that such back-projection can be performed even if there are multiple virtual views to be rendered; thus, the proposed CSR algorithm is applicable to situations where there exist multiple virtual viewpoints.

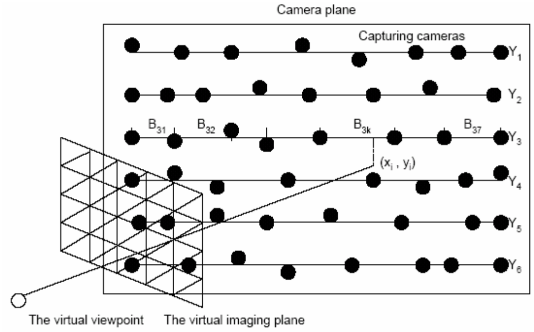


Fig. 8. Self-reconfiguration of the cameras.

3. *Collect the CCV 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 and k is the index of bins on that guide, $1 \le j \le 6$, $1 \le k \le 7$. If a back-projected vertex (x_i, y_i) satisfies

$$Y_{j-1} < y_i < Y_{j+1} \quad and \quad x_i \in B_{jk}, \tag{2}$$

the CCV 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 CCV scores for each linear guide, denoted as S_{jk}, where j is the row index of the linear guide and 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 $S_{jk}, 1 \le k \le 7$. 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 CCV 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]. The positions of the cameras during the movement are constantly monitored by the calibration process. After a fixed time period (400 ms), a ”stop” command will be issued to stop the camera motion.

7. End of epoch. Jump back to step 1.

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

B. Results

We show results of the proposed CSR algorithm in Figure 10. In Figure 10 (i) and (iii), the capturing cameras are evenly spaced on the linear guide. Figure 10(i) is rendered behind the camera plane, and Figure 10(iii) is rendered in front of the camera plane. Due to depth discontinuities, some artifacts can be observed in the rendered images (Figure 10 (i-d) and (iii-d)) along the object boundaries. Figure 10(b) is the reconstructed depth of the scene at the virtual viewpoint. Figure 10(c) is the CCV score obtained during the depth reconstruction. It is obvious that along the object boundaries, the CCV score is high, which usually means wrong/uncertain reconstructed depth, or bad rendering quality. The red dots in Figure 10(c) are the projections of the capturing camera positions to the virtual imaging plane.

Figure 10 (ii) and (iv) shows the rendering result after CSR. Figure 10 (ii) is the result after 6 epochs of camera movement, and Figure 10 (iv) is after 20 epochs. It can be seen from the CCV score map (Figure 10(c) that after the camera movement, the consistency generally gets better. The cameras have been moved, which is reflected as the red dots in 10(c). The cameras move toward the regions where the CCV score is high, which effectively increases the sampling rate for the rendering of those regions. Figure 10 (ii-d) and (iv-d) shows the rendering results after self-reconfiguration, which is much better than 10 (i-d) and (iii-d).

VII. Conclusions

We have presented a self-reconfigurable camera array in this paper. Our system is large scale (48 cameras), and has the unique characteristic that the cameras are mounted on mobile platforms. A real-time rendering algorithm was proposed, which is highly efficient and can be flexibly implemented in software. We also proposed a novel self-reconfiguration algorithm to move the cameras, and achieve better rendering quality than static camera arrays.

References

[1] H.-Y. Shum, S. B. Kang, and S.-C. Chan, "Survey of image-based representations and compression techniques," *IEEE Transaction on Circuit, System on Video Technology*, vol. 13, no. 11, pp. 1020–1037, 2003.

[2] 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.

[3] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan, "Image-based visual hulls," in *Proceedings of SIGGRAPH 2000*, ser. Computer Graphics Proceedings, Annual Conference Series, ACM. ACM Press / ACM SIGGRAPH, 2000, pp. 369–374.

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

[5] H. Schirmacher, M. Li, and H.-P. Seidel, "On-the-fly processing of generalized lumigraphs," in *EUROGRAPHICS 2001*, 2001.

[6] 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.

[7] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. Horowitz, "The light field video camera," in *Proceedings of Media Processors 2002*, ser. SPIE Electronic Imaging 2002, 2002.

[8] J. C. Yang, M. Everett, C. Buehler, and L. McMillan, "A real-time distributed light field camera," in *Eurographics Workshop on Rendering 2002*, 2002, pp. 1–10.

[9] T. Kanade, H. Saito, and S. Vedula, "The 3d room: Digitizing time-varying 3d events by synchronized multiple video streams," *Technical Report, CMU-RI-TR-98-34*, 1998.

[10] S. Vedula, "Image based spatio-temporal modeling and view interpolation of dynamic events," Ph.D. dissertation, Carnegie Mellon University, 2001.

[11] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong, "Plenoptic sampling," in *Proceedings of SIGGRAPH 2000*, ser. Computer Graphics Proceedings, Annual Conference Series, ACM. ACM Press / ACM SIGGRAPH, 2000, pp. 307–318.

[12] C. Zhang and T. Chen, "Spectral analysis for sampling image-based rendering data," *IEEE Transaction on Circuit, System on Video Technology*, vol. 13, no. 11, pp. 1038–1050, 2003.

[13] W. Matusik, C. Buehler, and L. McMillan, "Polyhedral visual hulls for real-time rendering," in *Proceedings of Eurographics Workshop on Rendering 2001*, 2001.

[14] B. Lok, "Online model reconstruction for interactive visual environments," in *Proc. Symposium on Interactive 3D Graphics 2001*, 2001.

[15] M. Li, M. Magnor, and H.-P. Seidel, "Hardware-accelerated visual hull reconstruction and rendering," in *Proc. of Graphics Interface 2003*, 2003.

[16] G. G. Slabaugh, R. W. Schafer, and M. C. Hans, "Image-based photo hulls," 2002.

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

[18] ——, "A system for active image-based rendering," in *IEEE Int. Conf. on Multimedia and Expo (ICME) 2004*, 2003.

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

[20] C. Zhang and T. Chen, "View-dependent non-uniform sampling for image-based rendering," in *IEEE Int. Conf. Image Processing (ICIP) 2004*, 2004.

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

[22] J.-Y. Bouguet, "Camera calibration toolbox for matlab," 1999.

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

[24] 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.

[25] C. Buehler, M. Bosse, L. McMillan, S. J. Gortler, and M. F. Cohen, "Unstructured lumigraph rendering," in *Proceedings of SIGGRAPH 2001*, ser. Computer Graphics Proceedings, Annual Conference Series, ACM. ACM Press / ACM SIGGRAPH, 2001, pp. 425–432.

[26] R. T. Collins, "A space-sweep approach to true multi-image matching," in *Proc. of CVPR '1996*, 1996.

[27] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," in *Proc. of CVPR '1997*, 1997.

[28] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, and N. Faust, "Real-time, continuous level of detail rendering of height fields," in *Proceedings of SIGGRAPH 1996*, ser. Computer Graphics Proceedings, Annual Conference Series, ACM. ACM Press / ACM SIGGRAPH, 1996, pp. 109–118.

[29] R. Szeliski, "Prediction error as a quality metric for motion and stereo," in *Proc. ICCV '1999*, 1999.

[30] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: Theory and experiment," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 16, no. 9, pp. 920–932, 1994.

[31] S. B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view stereo," in *Proc. CVPR '2001*, 2001.

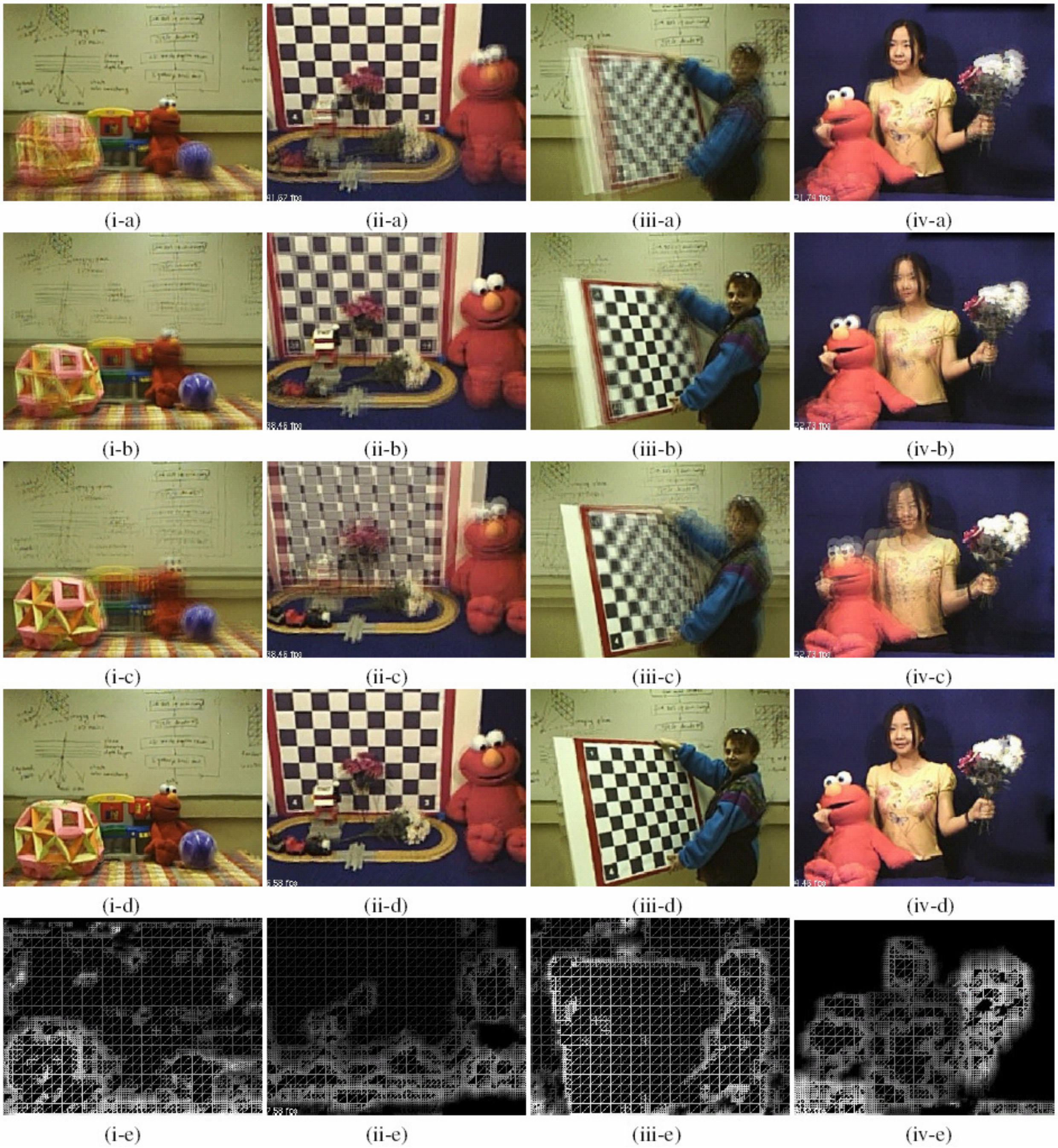
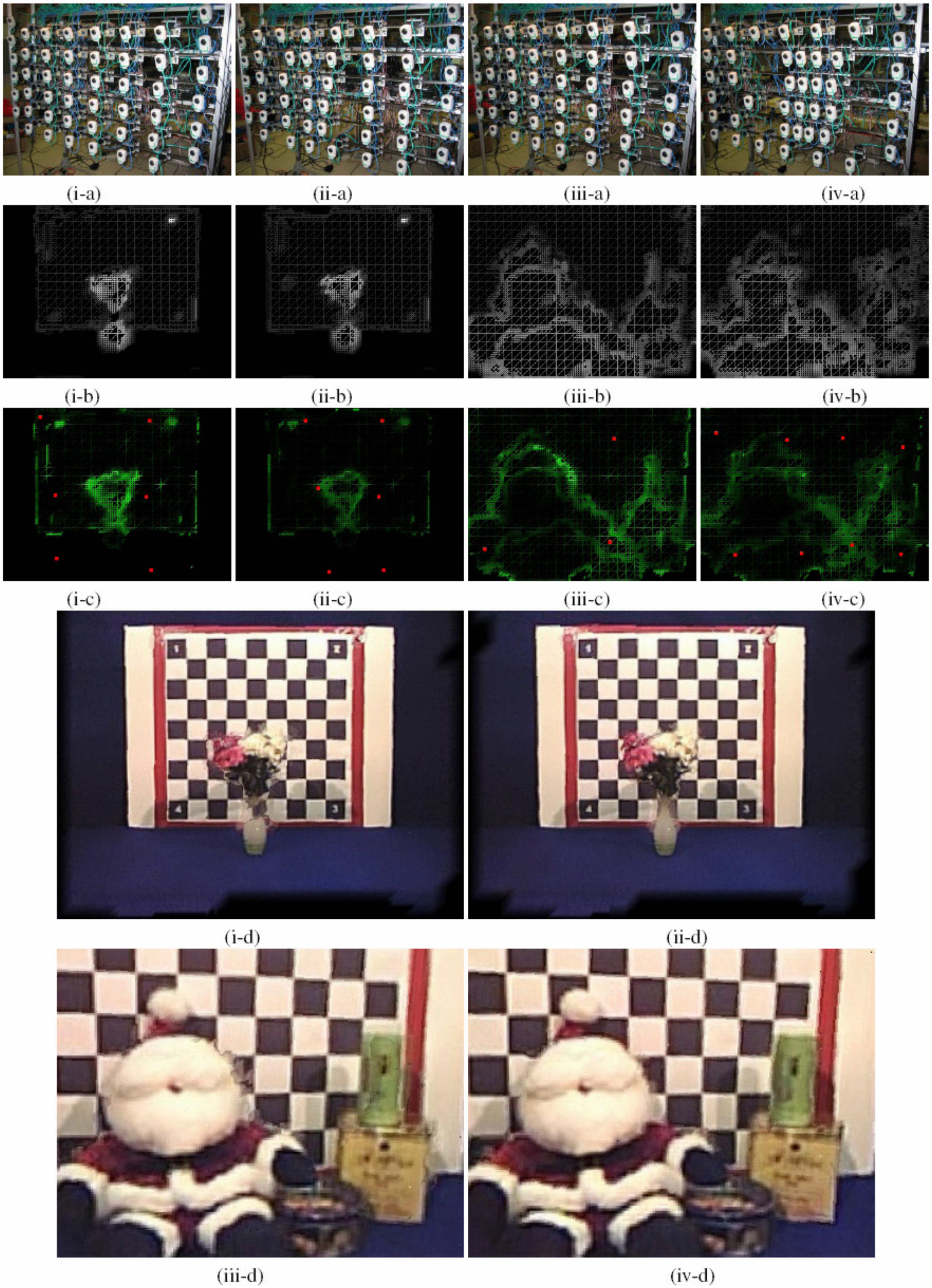


Fig. 9. Scenes captured and rendered with our camera array (no camera motion). (i) *Toys* scene. (ii) *Train* scene. (iii) *Girl and checkerboard* scene. (iv) *girl and flowers* scene. (a) Rendering with a constant depth at the background. (b) Rendering with a constant depth at the middle object. (c) Rendering with a constant depth at the closest object. (d) Rendering with the proposed method. (e) Multi-resolution 2D mesh with depth reconstructed on-the-fly. Brighter intensity means smaller depth.



(i-a) (ii-a) (iii-a) (iv-a)

(i-b) (ii-b) (iii-b) (iv-b)

(i-c) (ii-c) (iii-c) (iv-c)

(i-d) (ii-d)

(iii-d) (iv-d)

Fig. 10. Scenes rendered by reconfiguring our camera array. (i) *Flower* scene, cameras are evenly spaced. (ii) *Flower* scene, cameras are self-reconfigured (6 epochs). (iii) *Santa* scene, cameras are evenly spaced. (iv) *Santa* scene, cameras are self-reconfigured (20 epochs). (a) The camera arrangement. (b) Reconstructed depth map. Brighter intensity means smaller depth. (c) The CCV score of the mesh vertices and the projection of the camera positions to the virtual imaging plane (red dots). Darker intensity means better consistency. (d) Rendered image.