Image Based Routing for Image Based Rendering

Daniel Crispell
Brown University
Division of Engineering
daniel_crispell@brown.edu

Gabriel Taubin
Brown University
Division of Engineering
taubin@brown.edu

John Jannotti
Brown University
Department of Computer Science
jj@cs.brown.edu

***Abstract*— Recent research has shown that capturing a scene using several cameras has many advantages over traditional single viewpoint or stereo capture. Here, we focus on multi-camera systems using Internet Protocol (IP) as the mode of communication. As these networks of cameras grow in size, the problem of handling the immense amount of data they are capable of producing becomes an important problem. Our goal is to build a system consisting of many inexpensive cameras interconnected via an IP network and capable of rendering novel views in a distributed and bandwidth-minimizing fashion. The system should scale well as cameras are added to the network; we wish to design a system that is capable of handling on the order of 1000 cameras. The presented system utilizes in-network processing in order to minimize the amount of bandwidth needed. A novel data structure (the binmesh) is presented that enables us to efficiently aggregate representations of available views at each node. We present a working prototype of the system consisting of 16 cameras, as well as simulation results for networks of up to 1024 cameras.**

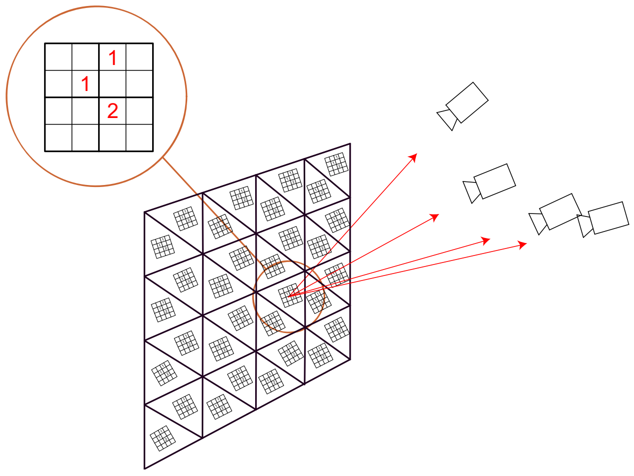


Fig. 1. The binmesh data structure - each face of the geometric proxy stores a set of bins which encode the directions to camera centers viewing the face.

I. Introduction

In recent years, there has been increasing interest in multiple camera systems [1], [2], [3]. As network-enabled cameras become smaller and less expensive, it is becoming feasible, physically and economically, to build systems consisting of networks of hundreds or even thousands of cameras. Although most, if not all, currently existing camera networks run on their own dedicated high speed links such as Firewire, we envision an environment where large numbers of network-enabled “smart” cameras are attached to arbitrary locations in a preexisting wired or wireless network and are able to be immediately integrated into a running image based rendering or tracking system. This type of camera network would have many potential uses, including security, surveillance, and entertainment. Towards this goal, we focus in this paper on designing and building an image based rendering system capable of synthesizing virtual camera views from an arbitrarily large number of networked cameras in a distributed fashion. Generated virtual camera views are useful for human interaction and display, as well as potential input to tracking algorithms and other in-network processing. We make no assumptions about the topography of the cameras, although it is assumed that each of the cameras intrinsic and extrinsic calibration parameters are obtainable by the camera. We wish to construct a system that is capable of handling large (1000+) numbers of cameras, while keeping network bandwidth usage and computation throughout the network to a minimum. We present a method for constructing such a system and some preliminary results.

When scaling multiple camera systems up, one obvious problem that quickly surfaces is an efficient way to handle the immense amount of data these systems are capable of producing. Even using VGA resolution (640x480) JPEG compressed image streams, a network of 100 cameras produces on the order of 1 Gbps of data (assuming 30 fps). All published work thus far has employed dedicated networks to handle this data. We wish to build our system on top of existing networks while disrupting preexisting traffic as little as possible. This constraint further limits the amount of bandwidth realistically at our disposal.

In order to achieve a system that scales well and minimizes needed bandwidth, it is clear that we need to aggregate redundant data when possible. When operating in an arbitrarily complex network, it is not practical for any one node in the network to know the locations, full paths, and views of all available cameras. Our goal is to minimize the amount of information stored and the amount of computation performed at any one node, as well as the total amount of data transferred over the network links. By dividing the rendered image into small fragments, we can route requests for these fragments to the cameras that will contribute to the rendering of the fragments only. The routing should be done in such a manner that it is not necessary for any one node in the system to maintain information about all other nodes. Additionally, we can aggregate replies (image data) to these requests at natural aggregation points in the network in order to save bandwidth and minimize the amount of processing done by the requesting node. Each node in the tree has to know four pieces of information:

1) a summary of available views from its children and their descendants
2) how to aggregate these views for presentation to its parent
3) how to distribute requests from its parent for a view that is available through one or more of its children
4) how to combine responses to these requests before propagating them back up to the parent

Our system accomplishes this with a tree based protocol conceived as a generalization of IP Multicast's [4] group join and forwarding mechanism. Smart cameras "subscribe" to requests for virtual views based on their fields of view. Virtual camera requests are routed only to those cameras that have indicated they might usefully fulfill the requests. Finally, image data is returned back through the dissemination tree, and aggregated along the way. This application-specific in-network processing might be provided by an active network [5] or an overlay network [6], [7]. In this paper, we present:

- a framework for a scalable camera network design using aggregation of camera view representation, data requests, and image data
- a data structure enabling efficient aggregation of camera viewpoints
- a working prototype and simulation of large-scale camera networks using this framework and data structure

A. *Language and terminology*

Throughout this paper, we refer to the camera network as consisting of a set of connected *nodes*, where a node can be either a router, a camera, or a host requesting views from a virtual camera. It is assumed for the purposes of this paper that routers are capable of being programmed and performing some simple image processing operations. Since cameras, like routers, are built using embedded processors, it is just as easy to imagine a system in which the cameras also have some programmability and can share the computational load. Indeed, we envision a future system in which the line between camera and router is blurred and each device in the network will be capable of image capture, image processing, and intelligent routing to other devices.

II. Previous work

A. *Multiple camera systems*

Early image based rendering systems developed by Levoy and Hanrahan [8] and Gortler et al. [9], [10], [11] produced novel views of a static scene using multiple images from a single moving camera. Light fields captured using a single camera have the obvious constraint of being limited to static scenes. In order to view a light field of video data, multiple capture devices are needed. Early work using multiple camera systems was proposed and presented by Kanade and Rander [12], [13] in their Virtualized Reality system. Several other multiple camera systems have been proposed and implemented since then, including the works of Yang [1], Naemura [14], Zitnick, [15], and Matsuik [2]. Recently, Wilburn et al. have demonstrated what is possible when camera systems are scaled up to 100 cameras [3]. Their cameras are connected via one or more Firewire busses, using a binary tree topology. Each bus is capable of handling up to 30 video streams simultaneously. The root nodes in the trees are connected to host PCs and write the video streams to hard disks. Although the cameras are physically connected using a tree topology, all communication takes place in a centralized manner from the host PC to each individual camera board.

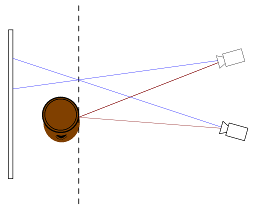


Fig. 2. When the focus plane (dashed line) is a poor approximation of actual scene depth, blended camera rays do not hit the same surface (blue rays), causing blurred images.

B. *Image based rendering*

Image based rendering has been an active area of research since 1996, when the Lumigraph [9] and Light Field Rendering [8] papers introduced systems capable of producing images of scenes from arbitrary viewpoints given a sufficient set of input images. This is accomplished conceptually by searching input images for pixels corresponding to rays that are as close as possible to rays corresponding to pixels of the desired view. Here, closeness can be defined in terms of intersection with the focus plane, angle difference, focus plane to camera distance, and other measures. The light field and lumigraph papers employed a "two plane parametrization" in order to represent the rays of both the input and desired views, where a ray is represented by its intersection points with a camera plane and a focus plane. One constraint of these algorithms is that the camera centers are assumed to be aligned to a regular planar grid. The lumigraph relaxes this constraint somewhat by employing "rebinning" of input image data, but this extra step causes a degradation of image quality [11].

Debevec et al. described a system [16] for rendering scenes with known geometry using view-dependant texture mapping. Each face of the scene geometry stored indexes corresponding to images in which the face was visible. Images from viewpoints similar to that of the virtual camera were used to texture map the faces. Although this system does not meet our requirements of scalability to large numbers of cameras (it is not meant to), we borrow the concept of view storage at the faces.

In 2001, Buehler et al. introduced a variation of the light field/lumigraph [11] that generalized away many of the constraints inherent in the original papers. The system makes no assumptions about the locations of the cameras, and can produce virtual views in realtime using commodity graphics hardware. The authors introduce the concept of a *geometric proxy*, or a set of vertices and faces that estimate the underlying 3D structure that is to be imaged. In the case where no previous knowledge of the structure is available (as is the case here), a simple planar mesh can be used. It should be noted that the further the geometry we wish to image is from the actual geometric proxy, the blurrier our output image will be due to the non-convergence of the blended camera rays on the object surface (Figure 2). The desired image is triangulated using a regular grid plus projections of geometric proxy vertices and projections of source camera centers. A weight for each camera is evaluated at the vertices of the triangulation, and each triangle is rendered once for each camera with a non-zero weighting at one of its vertices. The images are then blended based on the camera weights. Because this algorithm requires a centralized node that communicates directly with and performs computations based on each individual camera, it does not meet our goals.

Yang et al. introduced a system [1] that is capable of rendering virtual views using an eight by eight array of 64 Firewire video cameras in realtime. The centers of the source cameras are triangulated in 3D. These triangles are then projected onto the focal plane and rendered from the desired point of view using projective texture mapping. Each triangle is rendered once by each camera that corresponds to a vertex and blended by the compositor. It is assumed that the cameras have knowledge of their neighbors in the grid, and so can determine their individual contributions and weighting in the final image given only the desired viewpoint and focal plane. These parameters are not dependant on the source cameras, so it can be broadcast by the compositor to all cameras. The cameras themselves can then calculate their contribution (based on their position in the fixed topology) and return the weighted image fragments to the central node. Because it is assumed that the cameras are capable of transmitting their specific contributions only to the final image, the bandwidth used by the system is independent of the number of cameras and depends only on the number of views being rendered. While this is a very important result, it relies on an assumption that does not hold true in our desired environment: all cameras are arranged in a fixed, regular topology and that all cameras have knowledge of their neighbors in the topology. When this fixed regular topology assumption is violated, cameras can no longer determine their own weighting without knowledge of all other cameras in the system. This means that, although the texture mapping may still be performed in a distributed manner by the cameras, the routing and communication will take place in a centralized fashion, with direct knowledge of and communication links to all cameras from the central node. We would like to build a system consisting of an arbitrarily large number of cameras with completely unconstrained position both in the physical environment and the network. This makes direct knowledge of neighboring cameras impractical, since spatial locality does not necessarily imply network locality or even similar fields of views [17].

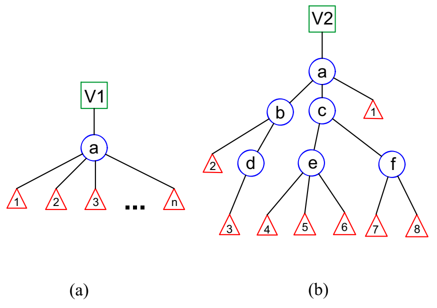


Fig. 3. (a) Previous distributed methods assume star-like network topology, with compositor (node a) located at center. The compositor must maintain information about all cameras 1,2,...n in this case. (b) Our method takes advantage of the actual (arbitrary) network topology. Node a maintains information about camera 1 and nodes b and c only, regardless of the underlying topology.

In general, the assumption made by all of the discussed multi-camera systems is that all cameras are directly connected to a central node (e.g. the compositor) in the network topology. Because the communication in a centralized system takes place via direct links between the central node and each of the source cameras, no aggregation of data along these paths is possible. Routing protocols such as IP enable the “direct link” abstraction to be made without loss of correctness, but greater efficiency and significant bandwidth savings can be achieved by performing processing at natural aggregation points in the network. We wish to scale to 1000s of cameras, so the centralized model is not practical.

In the case of image based rendering, aggregation of data flowing from the sources to the virtual camera is straightforward. If a node receives a face rendered by two or more of its children, the node can blend the fragments (taking into account their associated weightings) and propagate a single fragment to its parent node.

In addition to aggregating data on the path from the sources to the consumer, it would be advantageous to aggregate information about the cameras themselves. Ideally, we would like to avoid any one node (including the virtual camera) having to store explicit information about all of the cameras in the network. This has not been a problem in any existing implementations, but will become more impractical as the numbers of cameras grow. If information describing camera viewpoints can be aggregated, each node can store a representation of all views available over each of its links, and distribute incoming requests based on those representations. This aggregation of viewpoints, however, is less straightforward.

III. The Binmesh

The type and amount of image data that is useful varies greatly with the application. For example, if we wish to reconstruct the three dimensional structure of objects viewed by the cameras, it is necessary to have multiple (at least two

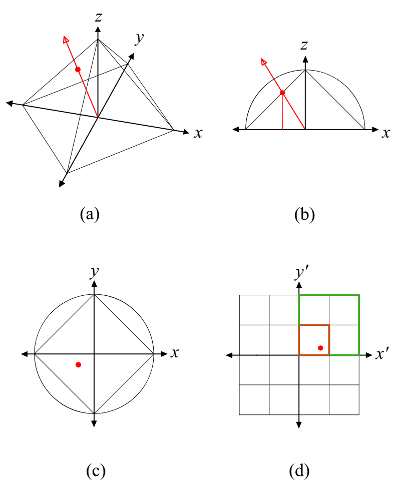


Fig. 4. (a) Unit pyramid (b) normal intersection with unit sphere and unit pyramid (c) unit pyramid orthographically projected onto xy plane (d) change of coordinates and first two subdivisions

but possibly many more) views of the same world points. This is also true of super-resolution applications. In the case of general surveillance however, we may want to minimize this redundancy and cover the maximum amount of the scene with as few images as possible. In general, it would be useful to have a compact representation of all available views of a scene and be able to make the decision at runtime which are transferred.

Borrowing the ideas of the geometric proxy [11] and the storage of camera view information in the scene geometry [16], we can begin to formulate a method for viewpoint aggregation in our camera network. We will render each face of the geometric proxy independently. The geometric proxy itself can be specified by the user or fixed if the scene geometry is known a priori. In our implementation, the geometric proxy is a planar mesh whose distance from the virtual camera is specified by the user. We would like to use views from cameras that are viewing each face of the proxy from as close an angle as possible to the angle at which the virtual camera is viewing it. Other factors may be taken into account as well (e.g. distance from the face) and weighted to give a general score to each virtual camera for each face. We make the assumption here that all considered cameras are viewing the focus plane from a comparable distance, so simply storing unit length vectors in the directions of the camera centers (from the face centroid) provides all the information we need in order to choose cameras for any virtual view of the face. In order to facilitate aggregation and selection, we use a novel method to store these vectors in a multigrid fashion (see Figure 4). The vector is first intersected with the "unit pyramid", or more precisely, the surface Π such that:

$$\Pi = \begin{cases} x+y+z = 1, & x,y>0 \\ x-y+z = 1, & x>0, y<0 \\ -x+y+z = 1, & x<0, y>0 \\ -x-y+z = 1, & x,y<0 \end{cases} \tag{1}$$

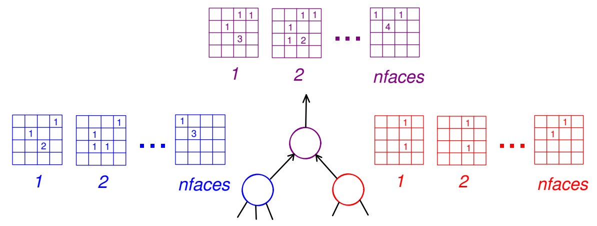


Fig. 5. Binmesh aggregation: vector bins of children are added, element by element, and the result is passed on to the parent node.

The z axis of the local coordinate system is the normal of the face. This allows us to restrict ourselves to storing vectors with positive z, since a negative value indicates a camera viewing the "back" side of the face. This intersection point is then projected orthographically onto the xy plane (i.e. the z component is simply dropped), and a change of coordinates is employed:

$$x' = x + y \tag{2}$$

$$y' = y - x \tag{3}$$

All unit vectors are thus mapped to a point on the xy plane within the unit square ($-1 <= x, y <= 1$). These mapped points are stored using a quadtree-like structure for efficient searching at multiple levels of angular resolution. Each face of our geometric proxy has one such vector bin associated with it. We call this structure consisting of a set of vertices, faces, and a camera angle bin for each face a *binmesh* (figure 1).

A. The data structure

Each vector bin is stored as an array with 4^n elements, where n is the number of subdivision levels used. Each subdivision is encoded with two bits such that we can efficiently recover any bin's four or eight-connected neighbors. By simply shifting a bin index two bits to the right, we can also obtain the index of the corresponding bin at the next level in the hierarchy. Each bin is stored using one byte, making this structure fairly large for reasonably fine-grained accuracy. In our implementation, we use six subdivision levels, meaning vector codes are 12 bits long and we must store 4096 bytes per face of the geometric proxy. Although these structures can become large for large proxies, they should be heavily compressible. Unless a system contains an extremely large number of well distributed cameras, each vector bin will be very sparse. In addition, neighboring faces in the geometric proxy will generally have similar vector bins. We emphasize that regardless of number and position of cameras that it represents, the entire uncompressed binmesh structure has a constant size for a given geometric proxy.

IV. Rendering Algorithm

A. Spanning tree construction

An important characteristic of the binmesh structure is that it is dependant only on the geometric proxy and the source camera positions, and not the virtual camera position. Because

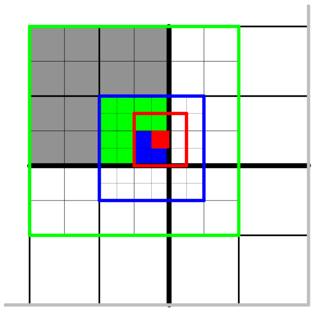


Fig. 6. Multiple resolution levels of the vector bins: Solid boxes represent the desired vector at multiple resolution levels. The outlined boxes show the searched eight-connected bins at each level. By searching a bin and its neighbors, we avoid the problem of missing matches lying across coarse level boundaries.

of this, we can use a spanning tree rooted at a router directly connected to the virtual camera node and update the tree only if a camera changes state, or our underlying geometric proxy changes. The virtual camera (or cameras) may move through the scene and receive different views of the geometry with no update of the structure. This rendering process begins with the injection of the geometric proxy into the network by the virtual camera node. The first node to receive this message becomes the root node in the spanning tree. The message is then propogated, hop by hop, to all nodes in the network with each node that receives the message (and is not already part of the tree) making the sending node its parent. When this process is completed, each internal node has one parent node, and one or more child nodes, which could be cameras or other routers. The root node has a direct connection to the virtual camera node. For each child, a router node creates a binmesh structure. In the case of a camera child, the node places a “1” in each appropriate bin of the binmesh. In the case of a router, the node passes the geometric proxy down and waits for a binmesh response. Once binmesh responses for all children have been received, all child binmeshes are aggregated and the result is reported to the parent node. Aggregation is accomplished with a simple element by element add of the binmeshes (Figure 5). This process continues until the virtual camera node has received a single binmesh representing the views of all available cameras in the network.

B. Novel view generation

When a virtual camera node wishes to generate a view, it sends a request message to the root node containing the extrinsic and intrinsic parameters of the virtual camera, as well as the set of visible faces of the geometric proxy to render. Each visible face index is accompanied by the vector to the virtual camera center from the face centroid, and the number k of actual views to blend in order to render the face. The request is then propagated to relevant cameras in the following manner: When a node receives a request, it chooses the set of children to redirect this request to based on the binmesh structures it has associated with each of them, and decrements k accordingly. For example, the root node may receive a request for a view of face f, to be generated using the best $k = 3$ views available. The root node may then request the view from child a, to be generated using $k = 2$ views, and child b, to be generated using the single best ($k = 1$) view only. How the requests are propagated is up to the individual router nodes. Router nodes may decide to give preference to cameras directly connected to the node and perform the mapping from the real camera to virtual camera themselves (thereby saving bandwidth), or may choose to give priority to children in order to pass off as much work as possible (load balancing may be achieved in this manner). In our implementation, the virtual camera requests that each face be rendered using the three best views. The nodes give preference to directly connected cameras in the case of equally good matches between camera and router children. The multigrid nature of the vector bins allows a node to first search at a coarse level and weed out children with zero or only poor matches, and then move to progressively finer levels of angular resolution until the desired number of matches is achieved.

A search at a particular resolution level consists of searching the desired bin and its eight-connected neighbors. We search in this way to avoid missing close matches that lie across coarse level bin boundaries (Figure 6). Cameras whose viewing directions closely match that of the virtual camera are weighted higher than others in the blending process. Closeness is determined by the finest level of resolution at which the source camera vector matches that of the virtual camera. Request replies from child nodes are received in the form of image fragments representing the virtual views of the requested faces. If a node requests (or directly maps) a face to more than one child, it must perform the blending of the two replies before propagating the fragment to its parent. The replies are blended according to their associated weights w_1 and w_2, and the composite fragment is given a new weight w_{new} based on the following equation:

$$w_{new} = w_{max} - \left[\frac{1}{(w_{max} - w_1)} + \frac{1}{(w_{max} - w_2)} \right]^{-1} \tag{4}$$

By assigning weights in such a manner, we ensure that fragments with equal individual weights will contribute to the composite fragment equally regardless of where in the network they are added. The more individual fragments a blended fragment is composed of, the heavier its weighting will be. The exception to these properties occurs if either of the weights w_1, w_2 are equal to w_{max}. In that case, w_{new} is assigned the value w_{max}. The blended fragments are then propagated to the parent node, and this process continues until the virtual camera node receives a reply containing all available fragments of the requested image. The virtual camera node has received a synthesized image, with the only computation required being the composition of the received fragments into a complete image.

V. Prototype Implementation

We have implemented a prototype of our system design using readily available IP webcams, a preexisting IP network,

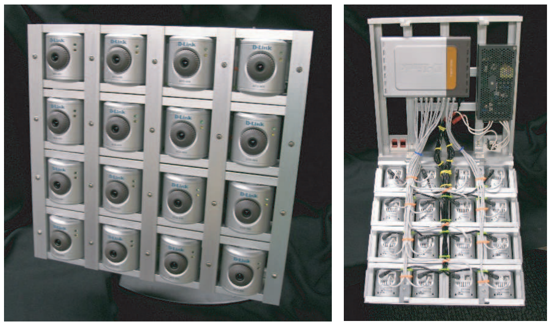


Fig. 7. (a) The four by four web camera array. (b) The 16 cameras are physically connected to a single router

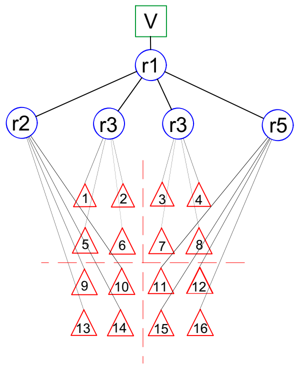


Fig. 8. Although the array cameras are physically connected to a single router, we simulate 5 routers, each with a fanout of four.

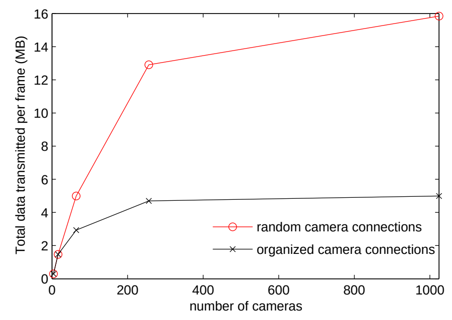
and PCs to perform the computations and routing. We do not currently have the capability of reprogramming either of the cameras or routers, so each router is simulated via an independent application running on one of the PCs, and is started with a configuration file indicating its directly connected camera and router nodes in the network. Using 16 of the cameras and a high speed switch, we have constructed a four by four camera array (Figure 7) for use in our system. The cameras are physically connected to a single switch, but we divide the array into four quadrants as shown in Figure 8 and simulate one router connected to each of four cameras in each quadrant. Using this setup, we are able to produce novel views in realtime (4 – 6 fps). Note that although all 16 cameras have a similar viewpoint because of the grid topology, this is not an assumption of the system. Any arrangement of cameras will work fine, including panoramic capture systems such as that presented by Swaminathan and Nayer [18].

A. Rendered image quality

Using the 16 camera array, two sets of images were rendered: One using a straightforward method similar to that presented by Buehler [11], and one using the distributed algorithm presented in this paper. In order to provide a ground truth for the rendering, a camera was removed from the array and a virtual view was rendered from that position. Figure 10 shows the true image, a result produced by the unstructured lumigraph implementation, and two views rendered using our method. The views are from the identical location but with different focus planes. The view rendered using the centralized algorithm appears slightly crisper, possibly due to the quantization of the normal vectors and the fact that in our implementation, camera distance is not taken into account when weighting the source images. Overall, however, our distributed algorithm produces comparable results to the centralized version.

VI. Large Network Simulation

In order to evaluate the systems capability to handle large numbers of cameras, we have simulated networks consisting of 4, 16, 64, 256, and 1024 cameras. In each simulation the cameras were arranged in a large, regular grid topology. We do not currently have image data corresponding to all of the cameras in the large networks, however if we make sure to render from viewpoints near clusters of cameras for which we do have data, the cameras with invalid image data do not contribute to the rendered image. The overlay trees used for view generation consisted of 1, 5, 85, and 341 router nodes, respectively. Each node in the tree had a fanout of four, with the cameras connected as leaves in the tree. The manner in which cameras are connected has a significant effect on the overall efficiency of the system, so we ran two sets of experiments: In the first set, we grouped the cameras according to spatial locality, as in Figure 8. In the second set of experiments, we chose the camera connections randomly, with no regard to their physical position. Using these simulated network topologies, we recorded the total amount of data being transferred over each of the links during the generation of a single frame from a virtual camera located within the boundaries of the grid. Figure 9 shows the results of these experiments. Because we use a tree structure and aggregate request and reply data at each node, the amount of total bandwidth needed will rise logarithmically with the total number of cameras in the network. Additionally, the bandwidth used over any given link is bounded by the size of a request/reply pair for the full set of geometric proxy faces (dashed line in Figure 9). If the overlay tree is organized in a spatially coherent manner such as that in Figure 8, and the number of cameras is much greater than the total number of “useful” cameras for a given view, we can gain further efficiency, roughly up to a constant factor of k. This is because at high levels of an “organized” tree, child links represent sets of cameras with very different views, so it is unlikely that the request will be propagated to more than one child. The best-case scenario occurs when a request for k fragments is directed to k cameras connected to the same router, since the data can be immediately aggregated at that node. The worst case occurs when the root node distributes a request to k distinct children, using roughly k times the bandwidth of the best-case scenario. This demonstrates the value of having a network topology that roughly corresponds to camera similarity: when the set of cameras that are useful for a given task are located near each



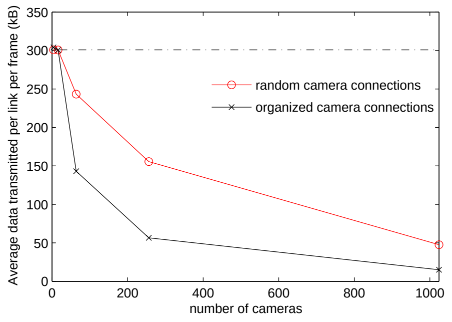


Fig. 9. Total and average per link bandwidth used in simulation. The dashed line represents the maximum possible data transferred over one link per frame (i.e. a request for all fragments of the proxy and its corresponding reply). The total bandwidth usage rises logarithmically with the number of cameras, and the efficiency of the system can be improved by connecting them in a coherent manner.

other in the network, data thinning is more effective and can be utilized at higher levels of the tree. In reality, our systems will fall somewhere between the two extreme cases presented in Figure 9; it is reasonable to assume that paths to nearby cameras will traverse many of the same links, however, for general camera topologies spatial locality does not necessarily imply similar views (e.g. orientations could be different).

The data presented assumes that the setup step described in Section IV-A has already been performed. This setup step requires the broadcast of the geometric proxy to all nodes, and the propagation of each nodes' binmesh structure to its parent. The size of the binmesh structure is dependant on the number of faces in the geometric proxy and the maximum resolution of the vector bins, but in general is relatively large. In our experiments, we used a planar geometric proxy with 338 faces, and six subdivision levels for the vector bins ($4^6 = 4096$ bytes per vector bin), leading to an uncompressed binmesh size of about 1.32 MB. For relatively small numbers of cameras or groups of cameras with similar viewing directions, these structures are very sparse and can be compressed before transfer if needed. Using gzip on its fastest setting, the binmesh of the root node in the 1024 camera simulation was compressed to under 21 kB (1.5% of the original). We do not perform any compression in the current implementation of our system.

VII. Future work

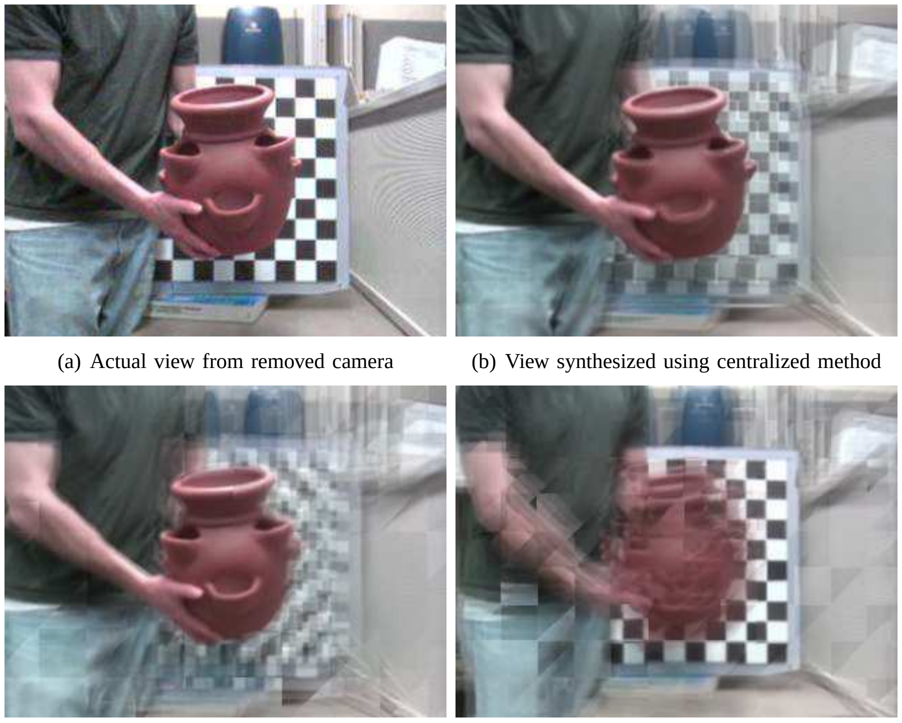
While these preliminary results are promising, some work has yet to be done to realize the true potential of the system. The first order of future work will involve building a system with more cameras. We plan to distribute many cameras throughout a conference room and experiment with the synthesization of views for remote collaboration purposes. In order to improve results for dynamic scenes, we will need to perform synchronization to assure that blended images correspond to temporally matching images. Several existing camera network implementations [3], [2], [13] use cameras with external trigger inputs to ensure exact synchronization. Our inexpensive webcams cannot be externally triggered, and further, wiring them to a trigger signal would violate our requirement that the cameras be interfaced to via the existing network only. Yang et al. [1] demonstrate a real-time system that achieves adequate frame correspondence by an initial synchronization of the internal clocks of the host PCs, and an agreed upon schedule for downloading images. By synchronizing the node clocks (using a protocol such as NTP) and specifying a download time with each frame request, we could implement a distributed version of such an algorithm. In addition to synchronization, we will need a more efficient way of calibrating our cameras. Currently, we calibrate each camera individually using Zhang's camera calibration technique [19]. Baker and Aloimonos [20] and Devarajan and Radke [17] have both presented methods for the calibration of multi-camera networks that would be better suited to our system as the number of cameras grows.

Another potentially interesting area to research could be the consolidation of data corresponding to multiple virtual camera requests. Currently, we treat each virtual camera independently, including the formation of the spanning tree. If we can assume that multiple virtual cameras share the same geometric proxy, we can save bandwidth by consolidating data corresponding to faces being viewed by multiple virtual cameras.

Currently, the geometric proxy is represented by a plane perpendicular to the virtual cameras viewing direction and at a user-specified distance. We would like to experiment with allowing the system to iteratively modify this proxy to achieve better focus at each face. Such a system would provide better looking pictures as well as an estimate of the underlying scene structure.

VIII. Conclusion

In this paper, we have presented preliminary results towards the implementation of a large-scale multi-camera system based on off the shelf components and IP networking technology. We demonstrate one possible use of such a network by generating novel views in a distributed fashion, using the binmesh structure to store available views in the network. We have compared the results produced using an implementation of our system with those produced by an existing centralized technique. We have also simulated the operation of our system using camera networks of various sizes, including large-scale



(a) Actual view from removed camera (b) View synthesized using centralized method

(c) Distributed method: focus plane in foreground (d) Distribured method: focus plane in background

Fig. 10. Ground truth image and rendered images using the 16 camera array

networks consisting of over 1000 cameras. Although this paper deals specifically with the problem of image based rendering, we view it as one instance of a more general problem and hope to apply the same principles of intelligent routing and distributed processing to other problems, including camera calibration and object tracking.

REFERENCES

[1] J. C. Yang, M. Everett, C. Buehler, and L. McMillan, "A real-time distributed light field camera," in *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, pp. 77–86.

[2] W. Matusik and H. Pfister, "3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 814–824, 2004.

[3] 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 *SIGGRAPH '05*, 2005.

[4] S. E. Deering, "Multicast routing in a datagram internetwork," Ph.D. dissertation, Stanford University, Dec. 1991.

[5] D. L. Tennenhouse, J. M. Smith, W. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[6] J. Touch, "Dynamic internet overlay deployment and management using the X-Bone," in *Computer Networks*, July 2001, pp. 117—135.

[7] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 131–145.

[8] M. Levoy and P. Hanrahan, "Light field rendering," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1996, pp. 31–42.

[9] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1996, pp. 43–54.

[10] A. Isaksen, L. McMillan, and S. J. Gortler, "Dynamically reparameterized light fields," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 297–306.

[11] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured lumigraph rendering," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 425–432.

[12] T. Kanade, P. Narayanan, and P. Rander, "Virtualized reality: Concepts and early results," in *IEEE Workshop on the Representation of Visual Scenes*, June 1995, pp. 69 – 76.

[13] P. Rander, P. Narayanan, and T. Kanade, "Virtualized reality: Constructing time-varying virtual worlds from real events," in *Proceedings of IEEE Visualization '97*, October 1997, pp. 277–283.

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

[15] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski, "High-quality video view interpolation using a layered representation," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 600–608, 2004.

[16] P. Debevec, Y. Yu, and G. Boshokov, "Efficient view-dependent image-based rendering with projective texture-mapping," Berkeley, CA, USA, Tech. Rep., 1998.

[17] D. Devarajan and R. Radke, "A distributed metric calibration of large camera networks," in *Broadnets 2004*, 2004.

[18] R. Swaminathan and S. Nayar, "Nonmetric calibration of wide-angle lenses and polycameras," *PAMI*, vol. 22, no. 10, pp. 1172–1178", October 2000.

[19] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations." in *ICCV*, 1999, pp. 666–673.

[20] P. Baker and Y. Aloimonos, "Calibration of a multicamera network," in *IEEE Workshop on Omnidirectional Vision*, 2003.