Real-time Mosaic for Multi-Camera Videoconferencing

Anton Klechenov, Aditya Kumar Gupta, Weng Fai Wong, Teck Khim Ng, Wee Kheng Leow
Singapore-MIT Alliance,
National University of Singapore,
Singapore

Abstract—**This paper describes a system for high resolution video conferencing. A number of camcorders are used to capture the video, which are then mosaiced to generate a wide angle panoramic view. Furthermore this system is made "real-time" by detecting changes and updating them on the mosaic. This system can be deployed on a single machine or on a cluster for better performance. It is also scalable and shows a good real-time performance. The main application for this system is videoconferencing for distance learning but it can be used for any high resolution broadcasting.**

***Index Terms*—real-time, mosaic, parallel, multi-camera**

I. Introduction

In present times distance learning is becoming more and more prevalent for education. Almost all top universities all over the world are making efforts to include telecommunication technologies into educational process. There are many different mediums in distance learning. Such as web pages, voice transmission, videoconferencing. It has been noted that the videoconferencing systems shows the best results [2], [7]. Knowledge absorption is very high using conferencing. So, when we speak about distance learning we usually mean videoconferencing.

Goal of our work is in development and improving advanced technologies for videoconferencing. Industry video camcorders usually provide very low resolution because they are oriented for use with off shelve TV sets. And the most reasonable way to increase resolution of video is by using several camcorders. But for using several camcorders we should solve some problems, such as cameras calibration, synchronization, video capturing and processing of several video streams in real time [1], [6].

We have addressed all these issues in our paper, we have developed a system that can accept streams of data from different pre-calibrated cameras and generate a single mosaic out of these different streams. Furthermore we have made our system "real-time" in the sense that we detect changes to the live mosaic and update it. This application thus gives the feeling of a single, virtual, wide angle camera instead of the array of cameras. Thus we can get a wide-angle real time panoramic video stream, which can be further transmitted in videoconferences.

We have described the hardware configurations used in the section 2, followed by the software infrastructure in the section 3. Following which we describe the algorithm and it's optimization in the section 4, results in section 5. Future work and conclusion is discussed in the sections 6 and 7.

II. Hardware Infrastructure

Industrial grade camcorders were used to develop our system. Currently we are using the camcorder from Sony corp. V500 series. These camcorders are connected to the PCs using the Firewire network. We also use a customized hub to connect more than two cameras to the PCs (PCs just have two ports for the Firewire network connections). We have used two configurations of the network: *serial* as shown in figure 1 and *parallel* in the figure 2. For the serial version of our system we have used a single PC with a network of camcorders. For the parallel version we have formed a simulated cluster of PCs. We say simulated cluster because it is not exactly a cluster, just a group of PCs connected using Ethernet. The windows implementation of MPI has a facility of setting certain environment variables on the PCs assigning each PC to be a certain node number in the cluster. Thus when we run MPI programs they run as if on a cluster (all the while they communicate with other nodes via the ether).

Manuscript received November 1, 2002. This work was supported by Singapore-MIT alliance.
Anton Klechenov (e-mail: antonkle@comp.nus.edu.sg).
Aditya Kumar Gupta (e-mail: smaakg@nus.edu.sg).
Wong Weng Fai (e-mail: wongwf@comp.nus.edu.sg).
Ng Teck Khim (e-mail: ngtk@comp.nus.edu.sg).
Leow Wee Kheng (e-mail: leowwk@comp.nus.edu.sg).

Serial configuration

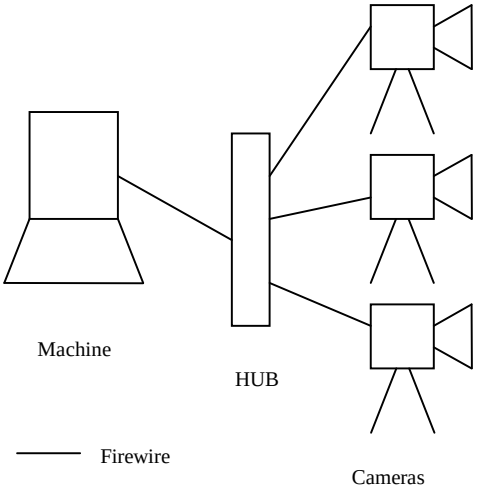

Figure 1: Machine and camera connected in the serial configuration

Parallel Configuration

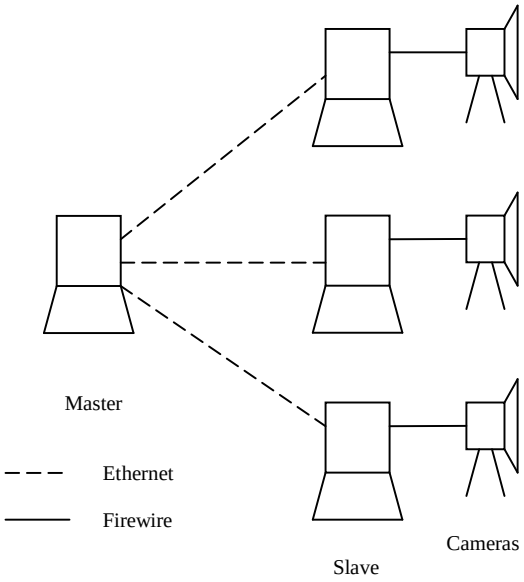

Figure 2: Machines and cameras connected in the parallel configuration

III. Software Infrastructure

For video capturing we use a C++ class library called the IC imaging control. This library lets us set the format of the video capture, adjust the brightness, contrast, etc…and set the frame rate on the machines. For the implementation of our algorithms we have used C++ programming with MS DirectX SDK and Intel OpenCV libraries. In the case of the parallel implementation we have used MPI (Message Passing Interface) programming model.

Software we used:

MS Windows XP;
MS VC++;
IC Imaging control;
MS DirectX;
Intel Performance Library;
Intel OpenCV library;
MPICH for MS Windows

IV. Algorithm

Below we define the algorithm of this system.

A. Camera calibration

Initially we need to set up the cameras to capture the scene and develop the mosaic. All camcorders we use are adjusted in such a way to have an overlapped area for each pair of them. In each overlapped area we have chessboard for camera calibration. The affine transformation parameters are obtained using the Kanade-Lucas-Tomasi [4] feature detector. This way, the process of cameras calibration is made automatic. Currently, 2D model of the scene is used. And for panoramic image affine transformation is suitable. But this approach works correctly only for the same depth object. In other words all objects of the scene are supposed to be at almost the same distance from the camera. Otherwise we have to build 3D models for the objects at the scene and affine transformation will not be suitable for mosaicing. For our current implementation difference in distance from camera to objects cause distortion on the panoramic image. In future we plan to add some stereo vision features for obtaining depth information.

B. Mosaic generation

After the camera calibration, we generate a mosaic of the screen where the subject in the video will appear [3], [5]. This would ideally be a blackboard, but for the purpose of calibrating the camera and generating the mosaic, we use some chess boards on the screen. Pairs of cameras will have a board that is overlapping and this will be used to generate the mosaic. This part has been implemented as a separate class, called BigMosaicing.

Mosaicing of complete set of cameras is done in two steps: first mosaic two adjacent camera images and apply the process iteratively to all the camera images until we get the complete mosaic.

1) Mosaicing Two Images

Mosaicing two images, image I and image I', involves several steps:

1. Identify corresponding point's p in I and p' in I'.
2. Compute affine transformation from p to p'.
3. Create resulting image R, for pixels p that exists only in I,

result R(p) = I(p).

4. For pixels p that are outside I but inside I', map p in I to p' = Ap in I'. Compute I'(p') using bilinear interpolation, and assign I'(p') to R(p).

5. For pixels p that are inside in the overlapping area of I and I', use a weighted average of both.

This program implements step 1 with two sub-steps. First, identify feature points in image I. We provide the method called findFeatures() in the Class BigMosaicing to find the feature points of an image. In the function, we call the routine provided in OpenCV (Intel Open Source Computer Vision Library) [10], cvGoodFeaturesToTrack() to identify the feature points. Secondly, find the corresponding points to those feature points of image I in image I'. Here we apply Lucas & Kanade Technique with pyramid approach, which is also implemented in OpenCV, called cvCalcOpticalFlowPyrLK(). We provide the method called findCorrespondingPointsSub() in the Class of BigMosaicing to identify corresponding points of two images. In this function, we use a modified version of the Lucas & Kanade Technique [8], [9]. Because we consider that Lucas & Kanade technique works only when the movement between the two images is small, this requires that the overlapping area of the two images must be very large. But in our application, it is meaningless if two images have a large overlapping area. So we assume that the two input images are in such a sequence that the first image's right part overlaps with the second image's left part, and that the overlapping area is within the half of the width of the two images. And since the two images are taken by the same type of camera, they have the same size. With this assumption, when we find the corresponding points of two images, we apply Lucas & Kanade Technique to the right half of the first image and the left half of the second image instead of the two original images so that the method can successfully find the corresponding points of the two images.

When we get the corresponding points of the two images, we compute the affine transformation of the two images with them. The method called getAffineTransform() in the Class of BigMosaicing calculates the affine transformation matrix from two sets of corresponding points.

With the affine transformation matrix of the two images, we can mosaic the two images as described above in step 3, 4 and 5. We implement it in the function named mosaicingGlobal() in the Class of BigMosaicing.

2) Combine Multiple Overlapped Images

The goal of the application is to combine multiple images taken by multiple cameras into a big image. Suppose there are n cameras, which take n images. When we combine the n images, we assume the n images are in the sequence that the i-th image overlaps with the (i-1)-th image on its left part, and overlaps with the (i+1)-th image on its right part. As mentioned in the previous part, the overlapping area with each other is within half width of the image.

When combining n images, we first mosaic the first image and the second image. Then mosaic the resulting image with the next image iteratively until we get the whole mosaicing result of the n images. We implemented it in the function named getBigMosaicing() in the Class of BigMosaicing.

Additionally, for each image, we provide a transformation matrix that transforms the pixel on the image to the corresponding pixel on the mosaiced result image. With the transformation, we can update the pixel on the mosaiced image which corresponds to the changed pixel of the original image. The function getMosPosition() of the Class BigMosaicing takes the pixel position of one of the original image, and returns the corresponding pixel position on the mosaicing image.

C. Real time updating

After we have the whole panoramic image we update it in real time. Each camera sends frames to central PC where position of each point is calculated using affine transformation matrix. We multiply coordinates of each point by transformation matrix to find the corresponding coordinates on the panoramic image.

For better performance we check the differences between consecutively appearing images from a camera and do change detection between the two images. From this it is possible to obtain the bounding rectangle of the changed area and update only this area in the mosaic.

D. Serial implementation

Serial version uses several video cameras connected to one PC using a FireWire hub. Here we found difficulties with frame rate of resulting mosaic video; the firewire network could not support the high resolution bandwidth which is need. Also, we found that updating is too slow in the serial version. Updating is done sequentially from all the cameras, thus the changes from camera one is updated, following which the changes from camera two and so on. This way there is a considerable delay in updating. Hence we had to resort to a parallel architecture for our system.

E. Parallel implementation

In the parallel configuration one PC is connected to each camera and one "central" PC for does the video mosaicing and updating. We tried configuration with one "master" PC for mosaicing and two "slave" PCs connected to two FireWire cameras. Connections between computers were built using different types of network.

We have implemented the parallel version using MPI programming on several MS Windows XP machines. We tried our parallel implementation using FireWire networking, 10 MBit Ethernet and 100 MBit Ethernet. Best results can be demonstrated with 100 MBit Ethernet. Performance of parallel system with 100 MBit Ethernet is much higher than serial version and other parallel versions.

We found difficulties with the FireWire network in the parallel implementation because of the bandwidth constraints in FireWire. The bandwidth of FireWire is 400Mb, but this network is transparent and this bandwidth is shared between all the cameras. When we try to send images from the cameras at the rate of 30 frames per second, the network over loads. Hence we use the FireWire network only to update the slave PC with the images from the cameras, but to transfer the updated from the slave to the master PCs, we use the Ethernet.

Parallel implementation of our approach for video mosaicing has an advantage - performance, because we can compute the changing rectangle on a "slave" PC and transmit information only within that rectangle to the "master" PC as a MPI message. On the "master" PC we should only update this changing rectangle. So, by using a parallel approach we have the possibility of increasing the number of cameras without a large decreasing in performance.

Each camera PC sends message to a central node. This is used for updating the rectangle on the panoramic image. This message contains the camera number, captured frame number, parameters of rectangle, such as coordinates and size, and array of points inside the rectangle. Size of the message depends on the size of the changing rectangle. So, every time “slave” node packs the data into a message and central PC unpacks it and updates the rectangle area on the result image.

F. Optimizations

First we implemented a version of video mosaicing for one PC connected to camcorders in a serial configuration. We found difficulties with frame rate of the resulting mosaicing video. Updating was too slow in serial version. For each updating, a frame from first camera is first updated, after that the frame from second camera and so on. And it takes some time to update frames from all the cameras. Frame rate for serial version with 2 cameras is only 6 frames per second. And it’s much slower for more cameras. We can use FireWire hub connect several camcorders to one PC. But the problem is performance. Updating is too slow and hence it is impossible to use serial version for videoconferencing. To address this problem we made a parallel implementation of program using cluster of PC’s. Another step for performance improving is transferring and updating not all frames but only changing regions.

G. Removal of artifacts/flickering

We eliminated image flickering in the overlapped area. The reason for flickering was updating overlapped area from both cameras. We calculate coordinates of overlapped area and update pixels in it only once using information obtained only from one camera. This simple procedure improved visual quality of resulting panoramic video.

V. RESULTS

We have measured the parameters in terms of the speed of the updates to the mosaic. We have chosen this parameter because this can give an estimate of the real time functioning of the system. We ran the tests under the two configurations of the system.

A. Serial implementation

In this configuration, we used 2 and 3 camcorders connected to a PC. This system gives a mosaic that is updated at a rate of 6 and 4 frames per second for 2 and 3 camcorders accordingly. This update is a little slow which is clearly visible. We find that the update rate decreases as we add more cameras to the system. This is because we have to update from multiple cameras and as this quantity increases, so does the overhead in updating.

B. Parallel implementation

In this configuration we used 2, 3 or 4 camcorders connected to PCs. Each of these slave machines were connected to the master machine. Here we got an update rate for the system as 12 frames per second. This shows significant improvements over the serial version.

Update rate for parallel configuration almost does not depend on number of cameras. It depends on size and speed of moving object at the scene. As far as we update only changing region value of update region is not a constant. If there are no changes at scene it will be no update at all. So, we measured frame rate for scene with moving object. It was one man, who walked near the wall all the time. For parallel version we can say that system working in real time and suitable for videoconferencing. Transferring and updating of only changing rectangle improved performance drastically. For parallel version it improves from 4 to 12 frames per second in average.

VI. FUTURE WORK

This application currently is in the form of a demonstration of the technology, this can be further enhanced. Currently, we have just developed the end part which generates the mosaic and does the real time change detection and updates it on a local machine. For the purpose of distance education it would be more practical to do the updating on a remote machine. Hence the part of system for data packing and transmitting has to be implemented. It is also possible to add some additional features like stereo vision for calculating depth of objects. For real tasks of videoconferencing lasers for camera calibration and mosaic parameters extraction can be useful. This would

make the initial setting of the mosaic more accurate. We see further development of our work in applying object tracking algorithms for panoramic video. It will allow us to track objects in field of view of several video cameras. Currently we use MPI for communication between slave nodes and master PC. In the future we planning to implement network module and use architecture client-server like in [6] for real time object tracking.

VII. CONCLUSION

In this paper we have described a system for real time, high resolution video generation. This has been done by doing an install time mosaic generation of the screen and subsequently doing a change generation. We have also added some optimizations to the system to make it run within acceptable delays. Furthermore we have deployed this system on a serial as well as a cluster of machines. And we see real time performance on the parallel version.

REFERENCES

[1] G. Kogut, M. Trivedi. Real-time wide area tracking: hardware and software infrastructure. IEEE CITS, September 2002.

[2] Y. Rui, A. Gupta, J. Grudin. Videography for telepresentations. Microsoft research technical report MSR-TR-2001-92, April 2002.

[3] R. Szelski. Video mosaic for virtual environments. IEEE Computer graphics and applications, 1996.

[4] J. Shi, C. Tomasi. Good features to track. IEEE CVPR, June 1994.

[5] A. Smolic, T. Wiegand. High-resolution video mosaicing. ICIP, 2001.

[6] X. Chen. Design of many-cameras tracking systems for scalability and efficient resource allocation. PhD thesis, Stanford University, July 2002.

[7] J. Cadiz, A. Balachandran, E. Sanocki, A. Gupta, J. Grudin, G. Jancke. Distance learning through distributed collaborative video viewing. Microsoft research technical report MSR-TR-2000-42, May 2000.

[8] C. Tomasi, T. Kanade. "Detection and tracking of point features", Technical report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[9] S. Birchfield, Derivation of Kanade-Lucas-Tomasi tracking equation, Unpublished, May 1996.

[10] Intel Open Source Computer Vision Library, http://www.intel.com/research/mrl/research/opencv