Monkeys — a Software Architecture for ViRoom — Low-Cost Multicamera System

Petr Doubek, Tomáš Svoboda, and Luc Van Gool

Computer Vision Laboratory, ETH Zürich, Switzerland

Abstract. This paper presents a software architecture for a software-synchronized multicamera setup. The software allows consistent multi-image acquisition, image processing and decision making. It consists of several programs that perform certain tasks and communicate with each other through shared memory space or TCP/IP packets. The software system can easily accommodate different number of FireWire digital cameras and networked computers. Minimal hardware requirement is the main advantage of the system which makes it flexible and transportable. The functionality of the system is demonstrated on a simple telepresence application.

1 Introduction

Smart multicamera systems are becoming more common due to the decreasing prices of powerful computers and digital cameras. Some of them were developed for creating realistic 3D models [4] or virtual views [5]. In late 1990's, new kinds of multicamera setups appeared. The main goal was not so much reconstruction but action recognition and interpretation. These applications mostly need real-time performance and current computers are capable of running wide range of image processing algorithms in real-time.

Cameras were often combined with microphone arrays for voice recognition and/or navigation. The EasyLiving project is developing an intelligent room that will be able to unobtrusively interact with a user [3]. The AVIARY project uses cameras and a microphone array for finding moving and/or speaking person. The detected speaker is then tracked and the best camera is chosen. A limited number of basic events are recognized [12]. Very recent project called M4 [1] tries to develop system that enables structuring, browsing and querying of an archive of analyzed meeting. In an addition to the projects creating whole environments, monocular methods are being extended to multiple views, especially tracking [7].

However, most of the systems are fixed to one room and they are still expensive for a casual user. Our goal is to develop a real-time multicamera system which would be flexible in terms of camera types and their number, easy to setup so that it can be moved from one place to another within hours, able to run on standard consumer hardware and affordable for almost anyone who already has a computer. The basic service that it should provide is a synchronized image capturing and the possibility to easily integrate both image processing algorithms working on single images as well as algorithms which use information from multiple cameras. Appropriate algorithms can be plugged-in for any particular application without the need to rebuild the whole system.

The mobility allows to move the system to the environment where the activities take place instead of moving the activities to the lab with the camera system. The production of training videos or the visual supervision of a machine maintenance in a factory are only two examples of applications of such a mobile multicamera system.

2 Hardware

We decided to use IEEE 1394 (FireWire) digital cameras which are now available in a large variety of types and prices. It is easy and cheap to add a FireWire card to a computer and new computers often already have FireWire ports included. Linux was chosen as an operating system for ViRoom because of its support for networking. We use several PCs connected by Ethernet and communicating over the TCP/IP protocol. Any Linux/PC computer connected to Internet can become a part of our system.

To keep the price of the whole setup low we use mainly simple webcams such as ADS[1] PYRO or Unibrain[2] Fire-i without precluding attaching higher quality cameras. Prices for the simple FireWire cameras start around 100 Euro which makes them affordable even for an occasional user or for home usage and it also allows to build systems with a high number of cameras. The main drawback is that they lack the external synchronization feature, so the software synchronization described have to be used. We describe it briefly in this paper, more details can be found in [11]. Although it is not as accurate solution as the hardware synchronization it requires no additional hardware or cables and thus helps to keep the system mobile and flexible.

3 Software Architecture

The overall software architecture is determined by the need for a software synchronization. The main control process, called Orangutan, synchronizes image capturing servers (CameraD) and collects the results from the image processing servers (Tamarin). The three processes mentioned above are necessary for any application and they will be described in this section. Inter-process communication and dataflow are explained in Fig. 1. Application-dependent multiple view algorithms are separate processes and they will be described in Sec. 4.

3.1 CameraD – Capturing Server

CameraD is a capturing server running on every computer with cameras attached. Its job is to wait for commands to capture images from the camera(s).

[1] http://www.adstech.com
[2] http://www.unibrain.com

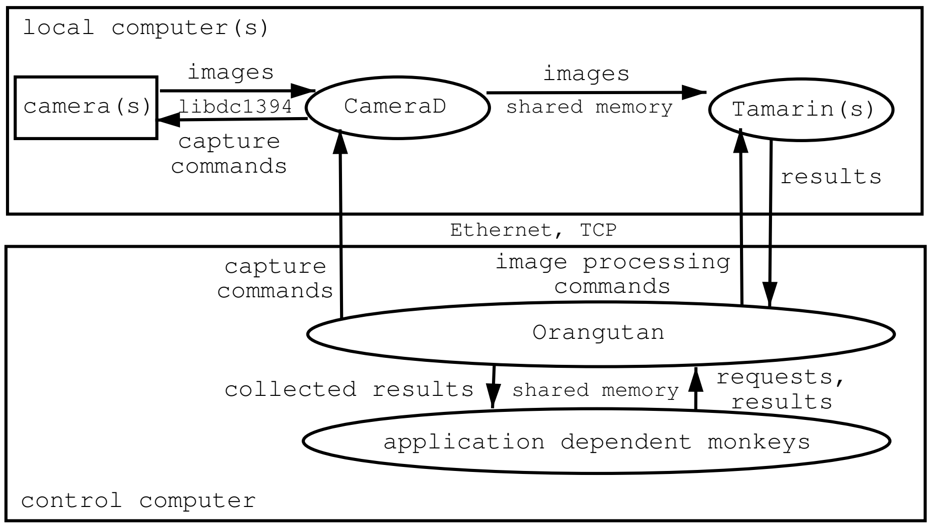


Fig. 1. Monkeys for the ViRoom. The top box shows processes running on the local computers with attached cameras. The bottom one shows the control computer with the control process. Arrows indicate commands and data exchanged between processes.

Captured images are stored in a cyclic buffer in a shared memory. This shared memory space is being accessed by other processes running on the same computer. Sending images over the network or saving them to the local disk are also possible options. Each stored image is identified by a camera number and a frame number. The frame number is a part of the capture command and serves for the time synchronization.

3.2 Tamarin – Image Processing Server

Tamarin is an image processing server running on computers with cameras attached. One Tamarin is started for each camera. It waits for commands to process images captured by CameraD. The frame number is attached to each command and Tamarin waits until the frame with this number appears in the shared memory. The commands for each frame are sent all at once, but results are sent back separately as soon as they are available.

The operations that Tamarin performs are supposed to be quite elementary. Currently image resampling, histogram calculation, background segmentation and silhouette extraction have been implemented. These operations may be combined to form more advanced tasks, e.g. sending extracted silhouettes from a segmented downsampled image together with a part of that image.

The required frame rate and output resolution or precision may vary for each application. Computationally expensive algorithms are performed on a downsampled image while the same frame in the original resolution can still be used for another task.

3.3 Orangutan – The Main Control Process

Orangutan is the main control process. It synchronizes CameraDs and collects the results from Tamarins. It parses a XML configuration file which specifies the desired application. The file contains instructions about processes to start, connection points (addresses, ports and shared memory keys) and commands to send, see Table 1 for an example. Its first task is to start all other processes including those running on remote computers. Orangutan passes the same configuration file to every process it starts so that they can establish necessary connections.

After this starting phase, it begins to send capturing commands to all CameraDs to synchronize them and immediately after that commands to the Tamarins to process the images. Then it gathers the results and stores them into shared memory. It does not run any algorithms on the results to ensure an undelayed sending of commands and gathering of results. Orangutan can run on any computer and we usually attach also one or two of the cameras to the computer with Orangutan.

```
<orangutan id="255" start_rid="1" shmkey="22000" period="66">

  <image start="yes" downscale="1"/>
  <lemur display_info="0" display_areas="1"/>

  <camerad id="2" name="viroom02" port="6666" shmkey="21005">
    <tamarin id="20" node="0" port="6700" mode="rgb"/>
    <tamarin id="21" node="1" port="6701" mode="rgb"/>
  </camerad>

</orangutan>
```

Table 1. An example of a simple configuration file. Orangutan starts the mentioned processes – CameraD and two Tamarins on host viroom02 and Lemur (will be discussed later) on the local computer. Tamarins will be asked to send downsampled images which will be stored locally by Orangutan and displayed by Lemur.

4 Multiple View Algorithms

The three processes described in Sec. 3 are necessary for any application. Their output is stored in the shared memory. We need also process(es) which display or further work out the results. The Lemur process has been developed for displaying. It can display images stored in the shared memory as well as other results, such as segmented images, silhouettes, histograms or virtual 3D objects etc. Algorithms using multiple image information run as another process and use also the same shared memory as Lemur for an input. Their results can be displayed, stored or they can pass requests back to Orangutan.

4.1 Gibbon – Closest View Selection for Telepresence

Gibbon is an example of a process working with information from multiple cameras. Its task is to select the closest view of a person moving in ViRoom. Teleteaching or teletraining are motivations for this specific application. The main goal is to provide an immersive video stream of an action that is being performed in a relatively large environment, a classroom or an assembly booth, for instance. One camera suffers from a limited and often occluded field of view. The main ViRoom idea is to place several cameras around or even inside the working volume and to let Gibbon keep the person optimally visible in the videostream throughout his or her action.

Gibbon runs on the same computer as Orangutan and has access to the shared memory where Orangutan stores results from the Tamarins, see Fig. 2. Tamarin runs background segmentation on each frame, based on [6] and [2]. Silhouettes are extracted from the segmented image using algorithm based on marching squares. Tamarin also computes the bounding box of the silhouette. The decision making process, Gibbon, selects the silhouette with the largest

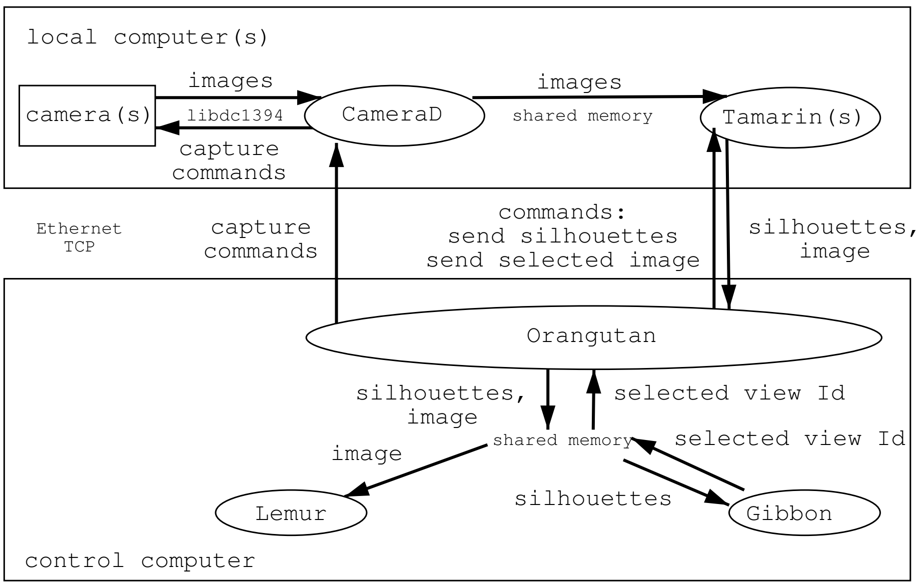


Fig. 2. Dataflow for the closest view selection application is shown in this figure. Compared to the general software architecture scheme in Fig. 1 there are two new processes on the control computer. Gibbon selects the closest view and Lemur displays it.

bounding box $s_{i,t}$ for each camera i and time frame t. A camera candidate for the best view is selected as $c_t = \arg\max_i area(s_{i,t})$. To prevent rapid and thus disturbing switching between views, when a person has about an equal size in two or more cameras, we set the switching resistance $r > 1$. To switch the view, the inequality $area(s_{c_t,t}) > r * area(s_{c_{t-1},t})$ has to be satisfied, otherwise c_t is set back to c_{t-1}.

Another feature offered by Lemur is virtual zooming. When it is turned on, Gibbon finds a rectangle inside the view c_t which contains the silhouette and has a specified aspect ratio. This information (selected view and zoom-in rectangle) is passed back to Orangutan. Orangutan then asks Tamarin to send only the requested rectangle from the next frame and Lemur will show the zoomed image of the person moving in ViRoom, see Fig. 3. The whole image is sent and displayed in the case that the virtual zoom is turned off.

It should be noted that this view selection is fairly simple and does not take into account changes in the background (for example a moved chair would be selected), or heading of the person (the closest view is not useful if we do not see the face). These drawbacks are currently being solved by the integration of a human tracker [9].

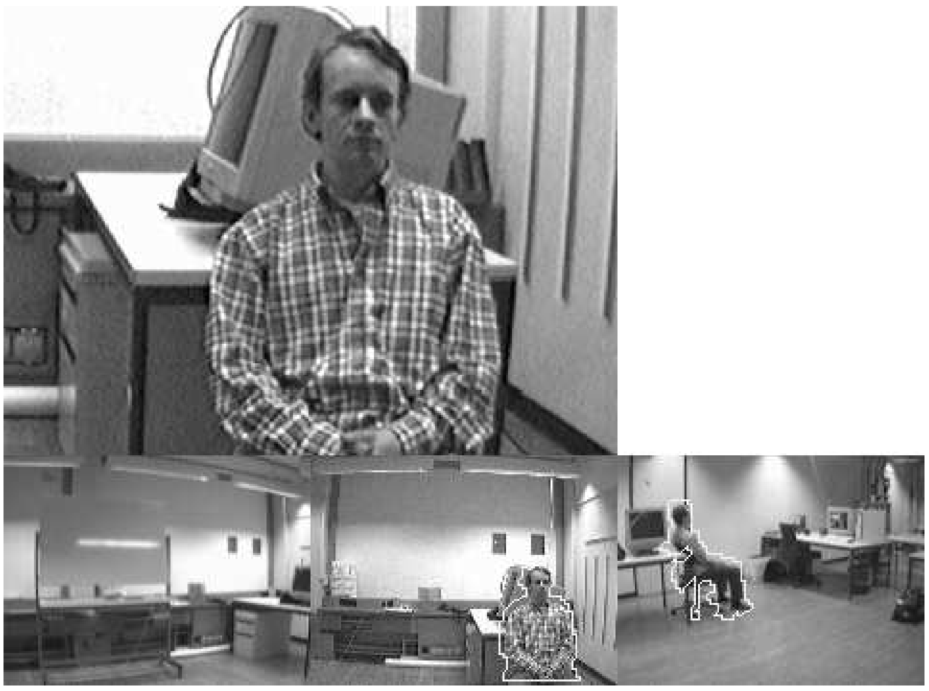


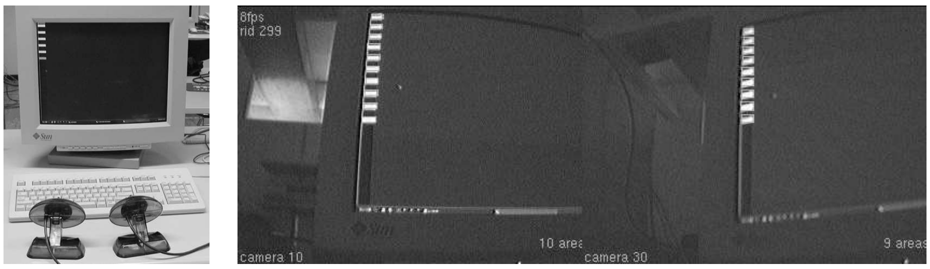
Fig. 3. Zoomed-in image of the segmented object is shown on the top, small images on the bottom line show available views and extracted silhouettes in them.

The application currently works with 320x240 pixel images which are down-sampled to 160x120 pixels for the segmentation and silhouette extraction because higher resolution silhouettes are not needed for this application. However, the selected view is sent in the original 320x240 resolution using JPEG compression. The image processing itself including the compression (which takes about half of the time) is fast enough to run at 30 frames per second, but because of the delays caused by the network and the capturing a framerate of 15 fps is used for a stable performance (on Pentium III 1 GHz processor). An alternative is to use 640x480 pixel resolution at 7.5 fps which presented in Fig. 3. There is space for improving the framerate or resolution by using faster compression, code optimizations or up-to-date CPUs.

4.2 Synchronization Error Measurement

In this experiment, the error of the software synchronization managed by Orangutan (see Sec. 3.3) is roughly measured for two cameras. The main limitations of this synchronization are network delay on a loaded network, latency of the Linux scheduler and unknown delays caused by camera drivers and hardware [11].

We point the cameras at a computer screen and display a simple box counter on the screen, see Fig. 4. The window on a screen is completely black at the start. Every 10 ms one white box is added in a line until they reach end of the screen and counting starts over. Tamarin segments the foreground white boxes, counts them and sends the result to Orangutan. The results from both Tamarins are stored in shared memory and compared against each other.



(a) (b)

Fig. 4. Synchronization error measurement. Both cameras (a) should capture at the same time and therefore see the same number of boxes (b). However, the left camera captured later and captured one box more compared to the right one.

This measurement is not entirely precise due to the limited refresh rate of the monitor (the refreshing period is 12-17 ms depending on settings used) and errors in the segmentation caused by noise. The length of a camera exposure is another limitation – during the exposure time box may light up which makes box count ambiguous. Nevertheless, calculating the difference between counted boxes for a large number of frames gives us idea how big the synchronization error is. An average value of about 20 ms was measured for typical frequencies we use (15 and 7.5 frames per second), which may be considered significant. However, our older experiments [11] showed that it is acceptable even for such a synchronization sensitive task as the self-calibration by a moving object.

5 System Setup

As already mentioned, the flexibility of the whole system is one of the main features. The system configuration is expected to change often by accommodating new cameras and computers or just simply changing the camera positions. Though many multiview algorithms may run with a totally uncalibrated system, the knowledge about its parameters may be very helpful or even necessary for many applications. Our geometric self-calibration and color alignment methods are sketched in the following two sections.

5.1 Self-Calibration

The theoretical basis can be found in [10]. Here, we describe briefly the practical realization. A person waves a standard laser pointer around the whole working volume. The very bright projections of the laser pointer can be detected in each image with subpixel precision by fitting an appropriate point spread function. These particular positions are then merged together over time, creating thus projections of a virtual 3D object. This virtual object is then projectively reconstructed. The final step is the Euclidean stratification that converts the projective reconstruction into Euclidean structures. More details can be found in [11]. The whole process runs without any user interaction, requires no special calibration object with known 3D coordinates and produces the complete linear camera models, ie. intrinsic and extrinsic parameters. All cameras are calibrated with respect to one common world frame.

5.2 Color Alignment

Having the same color characteristics across all cameras is necessary for a disturbance-free impression of the automatic camera selection algorithm described in Sec. 4.1. The color alignment would also ease correspondence search and color based tracking.

Our cameras have an auto-adjustment feature which changes gain, brightness and white balance dynamically. But this feature cannot be used, because it changes the camera parameters during the runtime without any notice to the software. An appearance of a large object in the camera view – for instance a person moving close to the camera – can cause a significant change in the parameters and affect the segmentation method.

The most precise way to align camera parameters would be to use some object with a known color for the calibration or use common parts of the scene and adjust the parameters to match their colors. Unfortunately, the use of some calibration object for the color adjustment is not suitable for a flexible and mobile system with such a large working volume.

Therefore, so called “medium gray world” is assumed and the algorithm described in [8] is applied. This algorithm changes the brightness, the gain and the white balance parameters of the camera to make the mean value in each color separation (R,G,B) equal to a predefined value (usually 50% of the maximal value), see Fig. 5. The parameters are adjusted for each view separately, because we do not have information about view overlap. We can either adjust the parameters each time the system starts or adjust them and save for later use.

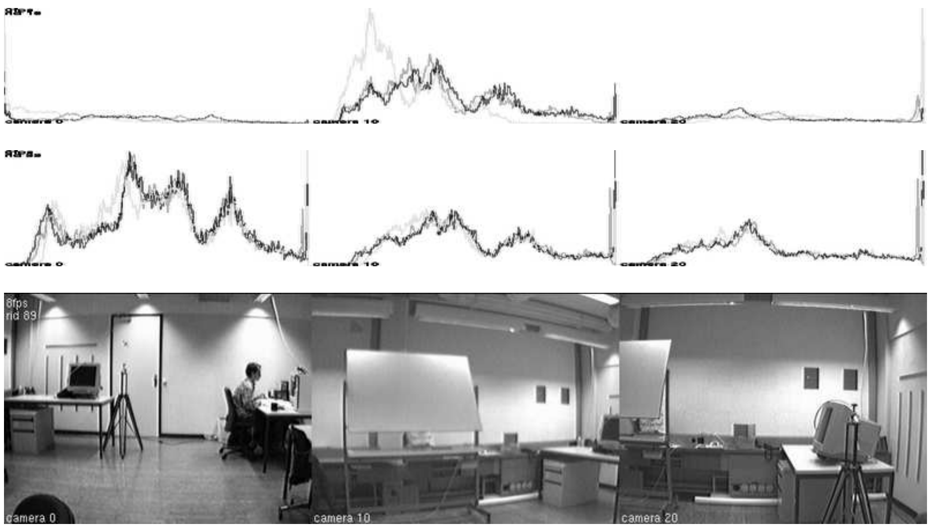


Fig. 5. Image color histograms before (top) and after (middle) the color alignment between three views. At the start the left image was too dark, the middle one lacks one color and the right one is too bright. The color alignment adjusts camera parameters so that colors are balanced and the mean intensity value is at 50% of the maximal value. The resulting images are shown on the bottom. A large number of high intensity pixels in the middle and left view is caused by the light reflected from white walls.

6 Conclusions

Monkeys for the ViRoom — the software architecture for software-synchronized multicamera system — were presented. This architecture allows to build low-cost systems for the applications where multiple views are essential. Moreover, it permits frequent reconfiguration, thus making the multicamera setup very flexible. We argue that the synchronization error — although it is high compared to hardware synchronization — is acceptable for a wide range of applications and we introduce the closest view selection for telepresence as a first application of the architecture.

The results were achieved with relatively simple image processing algorithms and the reliability of our self-calibration motivate us to upgrade our system by integrating new tracking algorithms such as [9] and by using 3D information. We are also working on improving reliability of the capturing software, especially in the case that more cameras are attached to one computer.

Acknowledgment. This work has been supported by ETH Zürich project BlueC and European Union project STAR.

References

1. M4 — multi-modal meeting manager. http://www.idiap.ch/˜mccowan/meeting/, Last visited on 25th October 2002. European project, 5th Framework Programme, IST.
2. Til Aach and André Kaup. Bayesian algorithmus for adaptive change detection in image seuences using Markov random fields. *Image Communication*, 7:147–160, 1995.
3. Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. Easyliving: Technologies for intelligent environments. In *Proceedings of the 2nd International Symposium on Hanheld and Ubiquitos Computing*, pages 12–29, September 2000.
4. Takeo Kanade, P.J. Narayanan, and Peter W. Rander. Virtualized reality: Concepts and early results. In *IEEE Workshop on the Representation of Visual Scenes*, pages 69–76, June 1995.
5. I. Kitahara, H. Saito, S. Akimichi, T. Onno, Y. Ohta, and T. Kanade. Large–scale virtualized reality. In *Computer Vision and Pattern Recognition, Technical Sketches*, June 2001.
6. Rudolf Mester, Til Aach, and Lutz Dümbgen. Illumination-invariant change detection using statistical colinearity criterion. In R. Radig and Florczyk S., editors, *DAGM2001*, number 2191 in LNCS, pages 170–177. Springer–Verlag, 2001.
7. Anurag Mittal and Larry S. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene using region-based stereo. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *The seventh European Conference on Computer Vision, ECCV2002*, volume 1 of *LNCS*, pages 18–36. Springer, May 2002.
8. Harsh Nanda and Ross Cutler. Practical calibrations for a real-time digital omnidirectional camera. In *Technical Sketches, Computer Vision and Pattern Recognition*, December 2001.
9. K. Nummiaro, E. B. Koller-Meier, and L. Van Gool. An adaptive color-based particle filter. *Image and Vision Computing*, page 3, 2002. To appear.
10. Mark Pollefeys, Reinhard Koch, and Luc Van Gool. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *International Journal of Computer Vision*, 32(1):7–25, August 1999.
11. Tomáš Svoboda, Hanspeter Hug, and Luc Van Gool. V**i**Room — low cost synchronized multicamera system and its self-calibration. In Luc Van Gool, editor, *Pattern Recognition, 24th DAGM Symposium*, number 2449 in LNCS, pages 515–522. Springer, September 2002.
12. Mohan M. Trivedi, Ivana Mikic, and Sailendra K. Bhonsle. Active camera networks and semantic event databases for intelligent environments. In *IEEE Workshop on Human Modeling, Analysis and Synthesis (in conjunction with CVPR)*, June 2000.