SIMULTANEOUS AND FAST 3D TRACKING OF MULTIPLE FACES IN VIDEO BY GPU-BASED STREAM PROCESSING

Oscar Mateo Lozano[*]*, Kazuhiro Otsuka*

omateo@locke.es, otsuka@eye.brl.ntt.co.jp
NTT Communication Science Laboratories
3-1, Morinosato-Wakamiya, Atsugi-shi, 243-0198, Japan

ABSTRACT

In this work, we implement a real-time visual tracker that targets the position and 3D pose of objects in video sequences, specifically faces. Using Stream Processors for performing the computations as well as efficient Sparse-Template-based particle filtering allows us to achieve real-time processing even when tracking multiple objects simultaneously in high-resolution video frames. Stream processing is a relatively new computing paradigm that permits the expression and execution of data-parallel algorithms with great efficiency and minimum effort. Using a GPU (Graphics Processing Unit, a consumer-grade Stream Processor) and the NVIDIA CUDA™ technology, we can achieve real-time performance even when tracking multiple objects in high-quality videos.

Index Terms— stream processing, GPGPU, particle filtering, video tracking, real-time systems

1. INTRODUCTION

Fast and robust object tracking in video sequences is required by many applications in many fields: automated surveillance systems need it to realize their objectives, robots rely on it to perform navigation tasks or man-machine interaction, augmented reality systems depend on the position data acquired by visual tracking to place their virtual objects in the real world, video-games or assisted devices can be controlled thanks to a camera and some face or hand tracking software, to name just a few.

Our motivation for developing a real-time face video tracker is to advance research on a system that can infer conversation structure from video sequences of face-to-face communication [1]; a key assumption is that the gaze direction of the participants provides cues for discerning the conversation structure, and can be identified from head directions. The constraints we impose on this tracker are: it has to be completely automatic, robust against rapid movement and partial occlusion, work with just one camera (no stereo-vision) on a conventional PC, and be able to track several faces simultaneously, all in real-time.

1.1. Particle filtering

Particle filtering is a model estimation technique based on Monte Carlo simulations [2]. Random values of a state-space variable are generated (the so-called particles), entered into a description of the system, and checked against the current output measure to generate a weight value, or probability of that particle being the one that best describes the current state of the system. Therefore, the collection of all these particles and their weights is, at each instant, a numerical approximation of the probability density function of the system. The Particle Filter (PF) framework is the basis of the well-known Condensation algorithm [3], which was originally proposed for contour tracking, but has been also successfully applied to the appearance-based tracking of moving objects in video sequences (as in our chosen method: Sparse Template Condensation [4]). The probabilistic approach of these methods provides significant robustness, as several possible states of the system are tracked at once at any given moment.

A common problem with this practical method is the significant computational requirements. Fortunately, Particle Filters are also easily parallelizable; while they require high arithmetic throughput, they have low global communication and storage costs. It is our belief that the advent of consumer-grade parallel processors can bring the robustness of these algorithms to real-time applications.

1.2. Stream Processing

We are specially interested in computer graphics chips (known as "Graphics Processing Units" or GPUs), because they are currently the most powerful, easily available, and cheap form of computing hardware. These chips have gone from fixed-application peripherals to modern, powerful, and programmable general purpose processors. In recent years, there has been strong interest from researchers and developers in exploiting the power of commodity graphics hardware for general-purpose computing (this movement is known as

[*]Author also with the Image Processing Group of the Universidad Politécnica de Madrid

Fig. 1. Results of the simultaneous tracking of four faces. The frame sequence is taken from a synthetic 1024x768 video, the sparse templates are composed of approximately 230 feature points, and each one is tracked using 1000 particles.

GPGPU, for "General Purpose GPU"). Unfortunately, the GPU uses an unusual programming model, so effective GPU programming requires rewriting the target algorithm into graphics terms by a programmer familiar with the limitations of the underlying hardware.

Stream processing represents an important advance in making parallel processing easily accessible to programmers. The programming paradigm raised by stream computing can be described as, given a set of input and output data (streams), to define some computation-intensive operations (kernel functions) that are to be applied to each element in the stream while exploiting the data independency and locality typical in media-processing applications. The programmer is forced by this (intuitive) programming model to express his/her application in a way that well suits the computational resources of Chip Multiprocessors (CMPs). But it offers more than just ease of programming: architectures that map well to this paradigm ("Stream Processors") can achieve higher performance than other architectures, as the locality and concurrency enforced by this paradigm (and the associated data bandwidth hierarchy) allows more of the die to be devoted to ALUs instead of caching and memory access circuitry. GPUs (although with some limitations) are considered to be general-purpose Stream Processors.

The Particle Filter framework, and specially the Sparse Template variant described in [4], applies particularly well to this paradigm: the operations performed are very simple (in our case, they consist mainly of geometric transformations), the data used is highly localized (each particle is self-contained, as also is every point that describes our template), and need little memory, so we can use the capabilities offered by Stream Processors (perform many computations extremely rapidly and in parallel) to achieve our real-time system. In particular, we expect a GPU to perform very well, and proof of this is that some authors have studied the application of pure GPGPU techniques to other particle filtering tracking algorithms with great success (like the 2D visual tracker in [5]). To the best of our knowledge, our work is the first 3D object tracker to be based on Stream processing.

2. METHOD OVERVIEW

Our tracker can be studied in three big blocks: initialization, tracking, and display. In our system, the initialization stage is performed in a separate thread in the host system, while the main thread performs GPU-based tracking and displays the results (Fig. 1).

2.1. Initialization stage

The initialization thread scans the image looking for new faces that are not currently being tracked. For this purpose, a

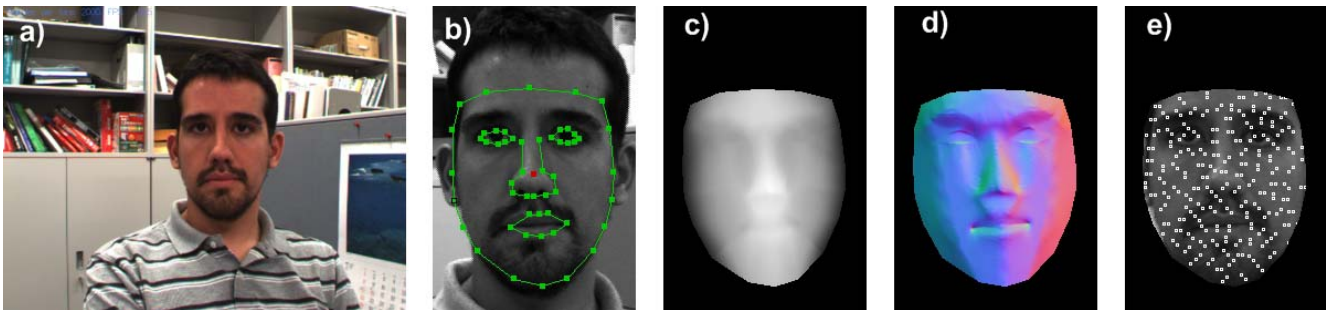

Fig. 2. **a** Frontal face image at time $t = 0$; **b** the Viola & Jones detector finds a rectangle containing a face, and the AAM is 2D-fitted to the shape of that face; **c** depth map texture is warped to the AAM shape; **d** the same for the normal map; **e** the feature points (position and grey level) that form the sparse template are selected using image processing techniques, and their depth and normal to the surface values are extracted from prior maps.

Viola & Jones [6] boosting algorithm is employed, as these detectors are quite fast and have been proven to work very well in practice. After determining that a detected face is not currently being tracked (by simply comparing it to known face positions), the subimage formed by the detected rectangle is passed to the next step, for template extraction.

In the Sparse Template Matching method, a sparse template is carefully formed by a small set of pixels (feature points) from a full template, with the idea of making the tracking more efficient by reducing the number of calculations. In addition, we resort to the Sparse Template Matching method to find relevant points to track, and treat those points as unidimensional streams of data.

In this method, the human face is typically approximated as a planar surface forming the sparse template. In order to increase precision, our system uses a generic 3D model of the human face that we personalize to each detected face by means of the Active Appearance Model [7]. We fit one of these models over the face subimage, thus obtaining the 2D coordinates of a series of landmark points that correspond to previously known features of the human face and, with the help of these landmarks, adapt our face model (described as the pair of a "heightmap" and a "normalmap") to obtain the depth value and normal to the surface vector at every point of the face. See Fig. 2 for a graphical explanation of this process.

Next, the feature points are selected from local minimum/maximum points and boundary dipoles on the image, as in [4]. With all this information, our template can be formed by a stream of N feature points, each one composed of the three coordinates of the feature point, and the vector normal to the face surface in the feature point. At the same time, M particles per object to be tracked are randomly selected and their state-space values filled with random values uniformly distributed around the well-known initial state. Here, a particle's state consists of 2-translations on the image plane, 3-rotations, scale, and illumination coefficient (See Fig. 1). The feature points and the particles provide our two input streams.

2.2. Weighting stage

Actual particle filtering is performed in this stage. As described in [2, 4], it consist of the steps of selection or drifting, diffusion, and particle weight measurement. Of all these steps, the last one is the most expensive. What this stage does is score each particle by means of a likelihood weight. Here, for each particle, the likelihood weight is calculated based on the summation of matching errors between each feature point in the template and corresponding pixels in the current frame. The particle filter algorithm itself is computationally expensive, but the weight computation is the main bottleneck, and the one that we have decided to execute via Stream processing. Our two input streams are composed of the stream formed by all particles and the stream formed by all feature points.

Weight calculation of each particle is an independent process, as is the matching error calculation for each feature point. Our method exploits these independencies to utilize parallel processing on the GPU: the kernels must perform the 3D transformation of each feature point as estimated by each of the particles, and then a comparison of the feature point gray level against the resulting point in the full image. The sum of all those comparisons for each feature point results in the weight of each one of the particles. This is our output stream: the collection of weight values of every particle.

As a simple feature point occlusion detector, we employ a normal map, as depicted in Fig. 2(d). It indicates the vectors normal to the surface in each feature point. By transforming (the same as we do with the feature positions, except only the rotation effect is considered) the normals, we obtain a coarse measure of the 3D face model pose, that we use to discard those points that are likely occluded by the face itself.

Once the weights of each particle are obtained, we rearrange the particles in descending order of weight and estimate the current state as the average of the best particles. Then, we pass the result to the display stage and perform the selection of new particles, and their random diffusion (by means of a

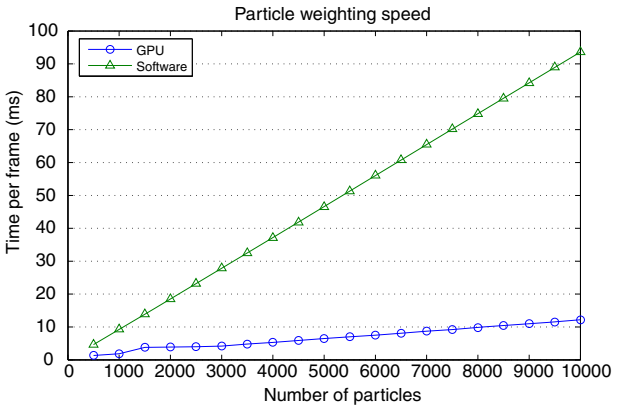

Fig. 3. Speed of the particle weighting stage: comparing Stream processing in the GPU version to a serial CPU-only version. Video 1024x768, 217 feature points.

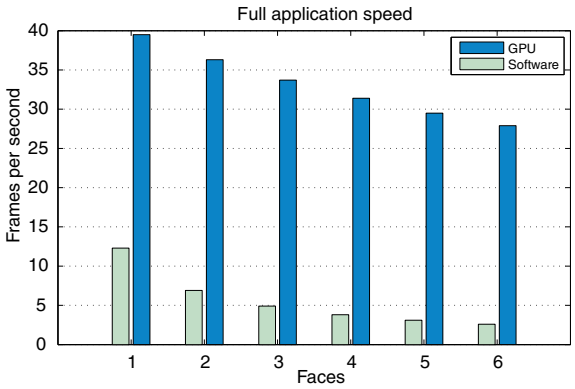

Fig. 4. Speed of the full application: comparing Stream processing in the GPU version to a serial CPU-only version. Video 1024x768, ≈230 feature points per face, and 1000 particles per face.

Gaussian noise term with mean and standard deviation chosen carefully to provide both stable and fast tracking) in order to diversify the set and avoid the degeneracy problem. After that, this stage starts again and is repeated indefinitely, unless the quality of the tracking results degrades so much that we must consider that tracking has failed.

3. EXPERIMENTS AND CONCLUSIONS

The developed software (a mixture of C++ and CUDA) was tested on an Intel Core 2 Duo 2.66GHz host system with 2GB RAM, using a NVIDIA GeForce 8800GTX GPU as the Stream Processor. As the adaptive face models, we used AAM-API [8]. The generic face model heightmap and normalmap were created by hand on the base of a subdivided CANDIDE [9]. The Viola & Jones implementation provided with OpenCV was used as the frontal face detector. The results indicate an important speed boost compared to the CPU-only version of the algorithm, especially when using a large number of particles (Fig. 3) and/or tracking multiple objects simultaneously (Fig. 4), making the tracker eminently suitable for real-time processing in a standard PC platform.

We have described a system for 3D visual tracking capable of achieving real-time performance thanks to the use of a GPU for parallel computation. The use of the Stream processing approach greatly simplified the development issues, and at the same time opened the door to other computing architectures. The goals imposed before starting the design (automatic, robust, just one camera, conventional computing resources, multi-object, real-time) have all been achieved, and the system is currently being used for future research in the area of conversation scene analysis. The novelty of the proposed work lies not only in the usage of a Stream Processor for 3D visual tracking, but also in the new sparse template initialization method that improves the accuracy and stability of tracking by means of a simple, generic 3D-model of the human face.

4. REFERENCES

[1] K. Otsuka, J. Yamato, Y. Takemae, and H. Murase, "Conversation scene analysis with dynamic Bayesian network based on visual head tracking," in *Proc. ICME2006*, 2006, pp. 949–952.

[2] A. Doucet, N. Freitas, and N. Gordon (eds), *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, 2001.

[3] M. Isard and A. Blake, "Condensation - conditional density propagation for visual tracking," in *Proc. ICCV'98*, 1998, pp. 107–112.

[4] Y. Matsubara and T. Shakunaga, "Sparse template matching and its application to real-time object tracking," *IPSJ Trans. Computer Vision and Image Media*, vol. 46, no. 9, pp. 17–40, 2005.

[5] A. S. Montemayor, J. J. Pantrigo, Á. Sánchez, and F. Fernández, "Particle filter on GPUs for real-time tracking," in *Proc. of ACM SIGGRAPH*, 2004, p. 94.

[6] P. Viola and M. Jones, "Robust real-time face detection," *IJCV*, vol. 57, no. 2, pp. 137–154, 2004.

[7] G. J. Edwards, C. J. Taylor, and T. F. Cootes, "Interpreting face images using active appearance models," in *Proc. Int. Conf. on Face and Gest. Recog.*, 1998, pp. 300–305.

[8] M. B. Stegmann, B. K. Ersbøll, and R. Larsen, "Fame - a flexible appearance modelling environment," *IEEE Trans. Med. Img.*, vol. 22, no. 10, pp. 1319–1331, 2003.

[9] J. Ahlberg, "Candide-3 – an updated parameterized face," Tech. Rep. LiTH-ISY-R-2326, Dept. Electrical Eng., Linköping Univ., 2001.