J Real-Time Image Proc (2012) 7:153–163
DOI 10.1007/s11554-011-0225-6

SPECIAL ISSUE

# A new parallel particle filter face tracking method based on heterogeneous system

Ke-Yan Liu · Yun-Hua Li · Shanqing Li ·
Liang Tang · Lei Wang

Received: 4 August 2009 / Accepted: 20 September 2011 / Published online: 21 October 2011

**Abstract** This paper proposed a multi-cue-based face-tracking algorithm with the supporting framework using parallel multi-core and one Graphic Processing Unit (GPU). Due to illumination and partial-occlusion problems, face tracking usually cannot stably work based on a single cue. Focusing on the above-mentioned problems, we first combined three different visual cues—color histogram, edge orientation histogram, and wavelet feature—under the framework of particle filters to considerably improve tracking performance. Furthermore, an online updating strategy made the algorithm adaptive to illumination changes and slight face rotations. Subsequently, attempting two parallel approaches resulted in real-time responses. However, the computational efficiency decreased considerably with the increase of particles and visual cues. In order to handle the large amount of computation costs resulting from the introduced multi-cue strategy, we explored two parallel computing techniques to speed up the tracking process, especially the most computation-intensive observational steps. One is a multi-core-based parallel algorithm with a MapReduce thread model, and the other is a GPU-based speedup approach. The GPU-based technique uses features-matching and particle weight computations, which have been put into the GPU kernel. The results demonstrate that the proposed face-tracking algorithm can work robustly with cluttered backgrounds and differing illuminations; the multi-core parallel scheme can increase the speed by 2–6 times compared with that of the corresponding sequential algorithms. Furthermore, a GPU parallel scheme and co-processing scheme can achieve a greater increase in speed (8×–12×) compared with the corresponding sequential algorithms.

**Keywords** Multi-core · Face tracking · Particle filter · General purpose computing on Graphic Processing Unit

K.-Y. Liu (✉)
Nokia-Siemens-Network CTO Research, Beijing, China
e-mail: keyan.liu@nsn.com

Y.-H. Li
Beihang University, Beijing, China
e-mail: yhli@buaa.edu.cn

S. Li
Institute of Scientific and Technical Information of China, Beijing, China
e-mail: lishanqing@istic.ac.cn

L. Tang
CETC No. 45 Research Institute, Beijing, China
e-mail: tangliang@45inst.com

L. Wang
HP Labs China, Beijing, China
e-mail: lei.wang13@hp.com

## 1 Introduction

Face detection and tracking is one of the most active research topics in computer vision and has made remarkable progress in decades of research. However, illumination and occlusion problems are still major challenges in face tracking. It is a promising approach to fuse multi-cue visual features to illumination changes. With high-performance hardware devices emerging, such as multi-core CPUs and GPU cards, face-tracking algorithms can be run more efficiently via parallel schemes, which make the fast and robust face-tracking algorithm useable in many applications, including video surveillance and human–computer interactions.

Multi-cues, such as color and contour [1, 2], color and motion cues [3, 4], and color and depth cues [5, 6], have been widely used to achieve robust object tracking. The corresponding experimental results show that the multi-cue-based approach improves tracking performance greatly. Compared with commonly used tracking methods, such as the Kalman filter [7] and mean shift [8], particle filter [9] can successfully solve the nonlinear and non-Gaussian problem at the cost of a high computation load. An object-tracking survey made by Yilmaz et al. [10] reviews the methods of object representation, feature selection, object detection, and object tracking in detail. Color, edge, optical flow, and texture are commonly used features for object tracking. However, the tracking performance is not robust using a single feature due to illumination variation, camera movement, and occlusions. A combination of these features is a promising approach for the improvement of tracking performance. This paper focuses on parallel particle-tracking methods with multi-cues.

Many researchers focus on establishing a multi-cue integration mechanism under the probabilistic framework, including the Dynamic Bayesian Network [11], the Monte Carlo method [1, 12], and particle filters [13–16]. In these methods, multiple cues are tightly coupled with the tracking model and the tracking algorithm based on a Bayesian framework, which make them difficult to use in deterministic tracking methods. Isard et al. [1] proposed an **ICondensation** algorithm combining color and contour information with importance sampling. Wu et al. [12] presented an approach to combine visual cues by including them in the state, but then decouple the prediction and observation of the different cues. Spengler and Schiele [13] proposed an integration scheme to reliably detect and track multiple hypotheses even under challenging conditions based on the Condensation algorithm. Maintaining multiple hypotheses over time explicitly avoids locking onto a particular target and therefore prevents an incorrect adaptation caused by false-positive tracking. Brasnett et al. [14] developed a particle filter (PF) and a Gaussian sum particle filter (GSPF) based on multiple information cues, namely color and texture, which are described with highly non-linear models. The algorithms rely on likelihood factorization as a product of the likelihoods of the cues. Serby et al. [15] presented a generic tracker combining complementary sources of information like interest points, edges, and homogeneous and textured regions for robust tracking performance. These features are integrated into a particle filter framework. Zhao et al. [16] proposed an effective and robust facial feature tracking approach based on the multi-cue particle filter. Both color and edge distributions were integrated into the filter to ensure tracking accuracy. An efficient updating algorithm was also introduced to avoid tracking error accumulation problems.

Another multi-cue-integration method is the pixel-wise integration method. In this method, tracking is considered to be a pixel classification problem. Whether a pixel belongs to the foreground or background is determined by all the cues. Every cue has a saliency map, and these maps are combined according to certain principles. One representative method is the adaptive democratic integration method proposed by Triesch and Malsburg [17]. Each cue votes for the final combined saliency maps, and the voting-like integration scheme is adaptive. Spengler and Schiele [13] used this adaptive integration method to integrate cues in human face tracking. This pixel-wise integration method is suitable for use in deterministic tracking methods.

Rasmussen et al. [18] defined a target as a conjunction of parts and introduced a constrained joint likelihood filter as a data-association method to generate the measurement for a Kalman filter. Chen et al. [19] used the Lucas–Kanade algorithm to detect and track six facial feature points using multiple cues, including facial feature intensity as well as the probability distribution, geometric characteristics, and motion information. Liu et al. [20] proposed an adaptive multi-cue integration for robust visual tracking based on the mean-shift framework.

Particle filter provides a statistical probabilistic framework to estimate object states based on sampling techniques. The observational step for calculating sample weights is the most computer-intensive stage in particle filter algorithms. As a result, a trade-off consideration is needed to make practical applications. Lozano and Otsuka [21] presented a GPU-based parallel scheme to create a real-time visual tracker of the position and a 3D pose of objects in video sequences. There are several parallel particle filter research papers on GPUs. Hendeby, Hol and Karlsson [22] conducted parallel particle filter research based on a particle resampling step. Montemayor et al. [23] implemented a real-time particle filter algorithm with the help of the shader model. Happe et al. [24] presented a video object-tracking application modeled on top of a framework for implementing SMC methods on CPU/FPGA-based systems, a heterogeneous multi-core architecture.

This paper proposed a multi-cue-based face-tracking algorithm with the help of parallel multi-core and GPU, which can effectively increase the robustness of the face-tracking algorithm. In Sect. 2, we briefly describe the multi-cue-based face-tracking algorithm within the particle filter framework, including the face model, dynamic model, and the multi-cue observation model. In Sect. 3, two parallel schemes for face tracking based on particle filter techniques were proposed to speed up the tracking algorithm. The results of the experiment in Sect. 4 demonstrate the effectiveness of our proposed algorithm. Finally, our conclusions are presented in Sect. 5.

## 2 Face tracking with particle filter

### 2.1 Tracking methods

Particle filter [9] offers a probabilistic framework for dynamic state estimation based on Monte Carlo simulations. This algorithm provides a robust tracking framework against the cluttered background. The state of a face at time $t$ is denoted $s_t$ and its history is $S = \{s_1, \cdots, s_t\}$. Similarly, the set of image features at time $t$ is $z_t$ with history $Z = \{z_1, \cdots, z_t\}$. The basic idea of particle filter algorithm is to compute the posterior state-density $p(s_t|z_t)$ at time $t$ using process density $p(s_t|s_{t-1})$and observation density $p(z_t|s_{t-1})$. To improve the face-tracking performance significantly, three cues of color, edge, and wavelet are integrated under the particle filtering framework. Given the face model, the tracking algorithm consists of four main steps: (1) sample selection, generating new samples $s_t'^{(n)}$ from old sample set $s_{t-1}'^{(n)}$ with its weights $\pi_{t-1}^{(n)}$; (2) prediction, determining new samples with dynamic model $s_t^{(n)} = s_t'^{(n)} + w_t^{(n)}$, where $w_t^{(n)}$ is the Gaussian noise at the $n$th iteration; (3) weight measurement, calculating the weights $\pi_t^{(n)}$ for each newly generated samples by observation steps; and (4) state estimation, obtaining the final state vector of a face by the newly generated samples and their weights. The implementation details are described as follows.

### 2.2 Face model

In this paper, multi-cues from a rectangular region are used to describe face features for tracking. As shown in Fig. 1, the red rectangle indicates the face region and three types of region features, including color histogram, edge orientation histogram, and wavelet features, which are

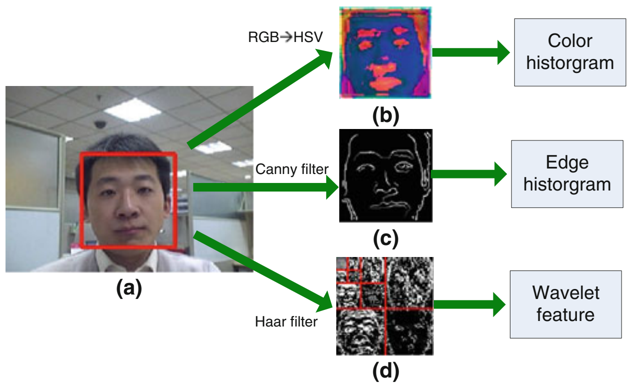


**Fig. 1** Face model feature description

integrated to achieve more stable tracking performance. To alleviate the problem of illumination change, we calculate the face color histogram $H^{\text{color}} = \{h_i^{\text{color}}\}_{i=0}^{B_c-1}$ in the HSV color space, where $B_c$ denotes the number of used bins (discrete intervals). Similarly, the edge orientation histogram $H^{\text{edge}} = \{h_i^{\text{edge}}\}_{i=0}^{B_e-1}$ is calculated by the edge image filtered by a Canny core, where $B_e$ denotes the number of used bins, as shown in Fig. 1c. For each face window, the horizontal, vertical and diagonal coefficients are calculated by wavelet transformations with different scales. Figure 1d indicates the final wavelet features $V^{\text{wavelet}} = \{v_i^{\text{wavelet}}\}_{i=0}^{d-1}$, where $d$ is the number of feature dimensions.

In our work, the face model is defined by the following parameters set as:

$$s = (H_{\text{color}}, H_{\text{edge}}, V_{\text{wavelet}}, R) \tag{1}$$

where $R$ is a rectangle represented by $R = (C_x, C_y, W, H)$, and $(C_x, C_y)$ is the centroid position, and $W$, $H$ are the width and height of the rectangle.

### 2.3 Dynamical and observation model

Face dynamics are modeled as a first-order process, as shown in the following equation:

$$s_t = s_{t-1} + w_{t-1} \tag{2}$$

where $w_t$ indicates the Gaussian noises. The observation process is performed to measure and weigh all the newly generated samples. The visual observation is a process of visual information fusion including three sub-processes: the computation sample weights $p^{\text{color}}$, $p^{\text{edge}}$, $p^{\text{wavelet}}$ based on color histogram, edge orientation histogram, and wavelet features, respectively.

For the $n$th sample, we obtained the weight $p_n^{\text{color}}$ through a Bhattacharyya similarity function as shown in Eq. 3, calculating the similarity between sample histogram $H_n^{\text{color}}$ and the reference histogram template $H_{\text{ref}}^{\text{color}}$.

$$p_n^{\text{color}} = \exp\{-D^2(H_n^{\text{color}}, H_{\text{ref}}^{\text{color}})\}$$
$$\text{where } D(H_n^{\text{color}}, H_{\text{ref}}^{\text{color}}) = \left(1 - \sum_{i=0}^{B_c-1} \sqrt{h_{i,n}^{\text{color}} \times h_{i,\text{ref}}^{\text{color}}}\right)^{1/2}. \tag{3}$$

Similarly, we obtain the sample weight $p_n^{\text{edge}}$ based on the edge orientation histogram as follows

$$p_n^{\text{edge}} = \exp\{-D^2(H_n^{\text{edge}}, H_{\text{ref}}^{\text{edge}})\}$$
$$\text{where } D(H_n^{\text{edge}}, H_{\text{ref}}^{\text{edge}}) = \left(1 - \sum_{i=0}^{B_e-1} \sqrt{h_{i,n}^{\text{edge}} \times h_{i,\text{ref}}^{\text{edge}}}\right)^{1/2}. \tag{4}$$

To compute the sample weight $p_n^{\text{wavelet}}$ based on the wavelet feature, the Euclidean distance between the sample feature vector $V_n^{\text{wavelet}}$ and the reference feature vector $V_{\text{ref}}^{\text{wavelet}}$ is employed in our system. The expression of $p_n^{\text{wavelet}}$ is as follows

$$p_n^{\text{wavelet}} = \exp\{-Eu(V_n^{\text{wavelet}}, V_{\text{ref}}^{\text{wavelet}})\}$$
$$\text{where } Eu(V_n^{\text{wavelet}}, V_{\text{ref}}^{\text{wavelet}}) = \left(\sum_{i=0}^{d-1} (v_{i,n}^{\text{wavelet}} - v_{i,\text{ref}}^{\text{wavelet}})^2\right)^{1/2}. \tag{5}$$

With three different visual cues, we obtain the final weight for the $n$th sample as:

$$p(z_t^n | s_t^n) = \alpha_{\text{color}} p_n^{\text{color}} + \alpha_{\text{edge}} p_n^{\text{edge}} + \alpha_{\text{wavelet}} p_n^{\text{wavelet}} \tag{6}$$

where $\alpha_{\text{color}}$, $\alpha_{\text{edge}}$ and $\alpha_{\text{wavelet}}$ are the coefficient values the weights of color histogram, edge orientation histogram, and wavelet feature cues, respectively. We can determine their values from experiences.

## 2.4 Multi-core particle filter algorithm

### *2.4.1 Initialization/re-initialization*

In most object-tracking systems, the initialization algorithm is only performed at the beginning of tracking and is incapable of recovering from tracking failures. In our tracking system, the boosted face detector [25] is introduced to achieve automatic initializations when the system starts or when a tracking failure occurs. The face detection results are used to update the reference face model. The updating criterion is confidence values that are less than a threshold value for $M$ successive frames. The online updating strategy can solve the tracking drift problems due to illumination changes, slight face rotations, or partial occlusions.

**Algorithm 1. Face tracking**

```
1) Automatic initialization / re-initialization
   A. Boosted face detection
   B. Reference face template updating
2) Particle filter tracking: probability density propagating from $\{(\mathbf{s}_{t-1}^{(n)}, \pi_{t-1}^{(n)}, c_{t-1}^{n})\}$ to $\{(\mathbf{s}_t^{(n)}, \pi_t^{(n)}, c_t^{n})\}$, where $c_t^n$
   indicates the cumulative weights for the n^th sample at time t.
   For $n = 1:N$
   A. Sample selection, generate a sample set $\mathbf{s}_t'^{(n)}$ as follows
      (a) Generate a random number $\alpha \in [0,1)$, uniformly distributed.
      (b) Find, by binary subdivision, the smallest j for which $c_{t-1}^{(n)} \geq \alpha$
      (c) Set $\mathbf{s}_t'^{(n)} = \mathbf{s}_{t-1}^{j}$
   B. Prediction, obtain $\mathbf{s}_t^{(n)}$ with $\mathbf{s}_t^{(n)} = \mathbf{s}_t'^{(n)} + w_t^n$
   C. Weight measurement
      (a) Observe with color histogram $p^{color} = \exp\{-D^2(H_n^{color}, H_{ref}^{color})\}$
      (b) Observe with edge orientation histogram $p^{edge} = \exp\{-D^2(H_n^{edge}, H_{ref}^{edge})\}$
      (c) Observe with wavelet feature $p^{wavelet} = \exp\{-D^2(V_n^{wavelet}, V_{ref}^{wavelet})\}$
      (d) Calculate the sample weight $p(z_t^n \mid s_t^n) = \alpha_{color} p_n^{color} + \alpha_{edge} p_n^{edge} + \alpha_{walet} p_n^{wavelet}$
   End
3) Normalize the sample set so that $\sum_n \pi_t^{(n)} = 1$, and update the $c_t^{(n)} \geq \alpha$ as follows
   $c_t^{(n)} = c_t^{(n-1)} + \pi_t^n$, $c_t^{(0)} = 0$,
4) Estimate state parameters of the pointing gesture at time-step $t$: $\hat{s} = \sum_{i=1}^{N} \pi_t^{(i)} s_t^{(i)} \Big/ \sum_{i=1}^{N} \pi_t^{(i)}$
5) If updating criteria are satisfied, go to step 1); otherwise, go to step 2).
```

### 2.4.2 *Algorithms*

The face-tracking algorithm consists of two parts: automatic initialization and particle filter tracking. Here, the algorithm is summarized, and is called Algorithm 1.

## 3 Parallel particle filter algorithm

### 3.1 Multi-core and GPGPU

There is a growing trend towards the use of multi-core chips. Dual-core and quad-core chips are very common, with many higher multiple-core chips to come in the future. Multi-core processors can deliver significant performance benefits for multi-threaded software by adding processing power with minimal latency, given the proximity of the processors. New multi-core processors will eventually come in heterogeneous configurations such as combinations of high- and low-power cores, graphical processors, cache blocks, and on-chip interconnects. By bundling many CPUs into one processor, creating what is known as a "manycore processor", the computing industry can continue to provide dramatic increases in computing power. As shown in Fig. 2a, the dual-core CPU has two CPU-local level 1 caches, and a shared, on-die level 2 cache (http://en.wikipedia.org/wiki/Multi-core).

Recent graphics architectures provide tremendous memory bandwidth and computational horsepower. One emerging trend is that of GPGPU. Packed with 340 Gflops, GPGPUs have been used to speed up many applications other than graphics processing. GPU processing power has been growing much faster than that of CPUs (multi-core) recently, and the trend continues. Other examples in this family include Cell Architecture, which has been used in PlayStation III, workstations, and servers from Mercury Computer Systems as well as the System-on-a-Chip from AMD for laptop computers, in which a GPU is integrated within the CPU. Three major advances in the last 2 years have made GPUs much more accessible to general-purpose computing: unified architecture design, extended C programming interface, and libraries and demonstrations of several applications.

The G80 processor, NVIDIA's first implementation of Compute Unified Device Architecture (CUDA), is an attempt at making GPGPU available for non-vector rendering programming. The G80 architecture enables the GPU to use multiprocessors to process blocks of 64–512 threads. The blocks are divided into groups of 32 called warps and are used by the processor for scheduling. The kernel is a program, executed as blocks of warps. The arrangement of threads in blocks and the blocks into grids of blocks is defined by the programmers, as shown in Fig. 2b.

### 3.2 Multi-core parallel particle filter algorithm

Traditional parallel/distributed programming techniques, such as message passing and shared-memory threads, are too complex for most researchers and developers, especially those without formal training on parallel and distributed computing.

MapReduce (http://en.wikipedia.org/wiki/MapReduce) provides a programming model for processing and generating large datasets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function merges all intermediate values associated with the same intermediate key. Programmers do not need to undertake the datasets parallelizing process, and the programs written in the functional style are automatically parallelized and executed on multi-

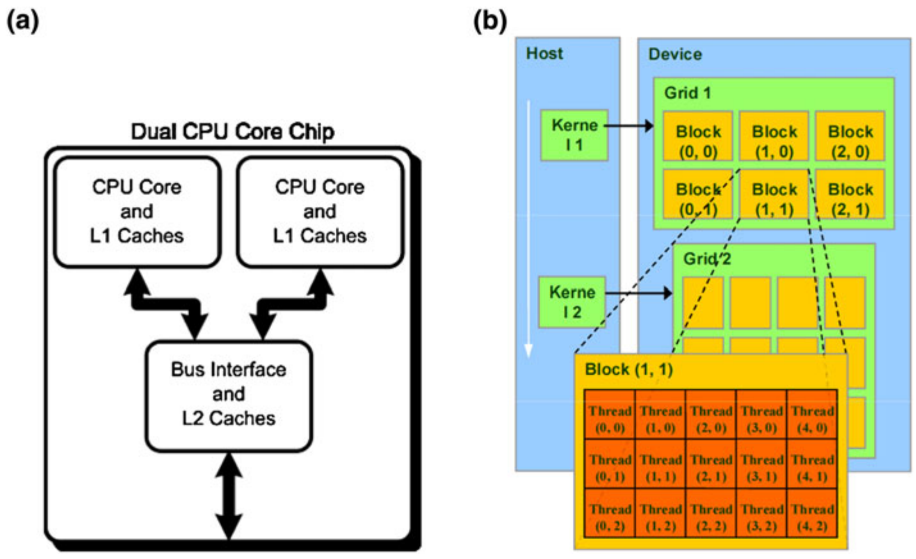


**Fig. 2** Multi-core and GPGPU thread structure (http://en.wikipedia.org/wiki/Multi-core). **a** Diagram of a generic dual-core processor. **b** NVIDIA G80 thread execution model (http://www.nvidia.com/cuda)

core machines and multi-core clusters. As parallel processing at the multi-core level is easier and more effective, we will attempt to use multi-cores in the particle filter algorithm.

Google's MapReduce is designed with their typical application in mind—that is, a dataset too large for the memory—and thus, a distributed file system is needed. Recent work at Stanford University has also proven that MapReduce interfaces and libraries can be good parallel programming paradigms for Symmetrical Multi-Processing (SMP).

The particle filter algorithm in face tracking can be classified into two levels. One is a multi-core implementation with the MapReduce model (multi-threading). Another level assigns the weight measuring portion to the GPU. We would like to describe our multi-core MapReduce thread model firstly. The parallel part is in the particle filter tracking iteration. If the particle number can be expressed as *particle_num* and the max number of available cores is *max_core*, the particle number on every core can be formulated as *particleNum_on_everyCore* = *particle_num/max_core*. The thread model for the particle filter algorithm can be shown in Fig. 3, in which the particle filter number is portioned with the current image (frame). Every thread on every core is bound to implement one map. The map procedure comprises the color histogram calculation, edge histogram calculation, texture feature calculation, and the maximum confidence. The maximum confidence in a thread is based on the three individual probability computations. The reduce procedure makes an aggregation with the different particle filters in the current frame.

## 3.3 Parallel particle filter algorithm on GPU

The most computationally consuming step is to calculate the weight measurements for all samples, especially with multi-cue features. The weight measurements for all samples are independent of each other. This character makes it easy to speed up the tracking algorithm with parallel computing hardware. The GPGPU, a parallel computing technique, can be applied to solve this problem.

The kernel for the GPU is executed in $M$ blocks, each with $N$ threads. Each thread calculates the weight measurement for one sample, including three measurements from the color, edge and wavelet features, respectively. Consequently, $M \times N$ samples can be processed in a parallel way. The parallel-tracking scheme is shown in Fig. 4.

As shown in the Fig. 4, the particle information corresponding to all faces and frame information are copied to the GPU memory for each input frame. Then the kernel function is invoked and is executed over the blocks and threads. Each thread calculates the color cue match, edge match and wavelet cue match for each particle. Color cue match can be taken as an example to explain the execution

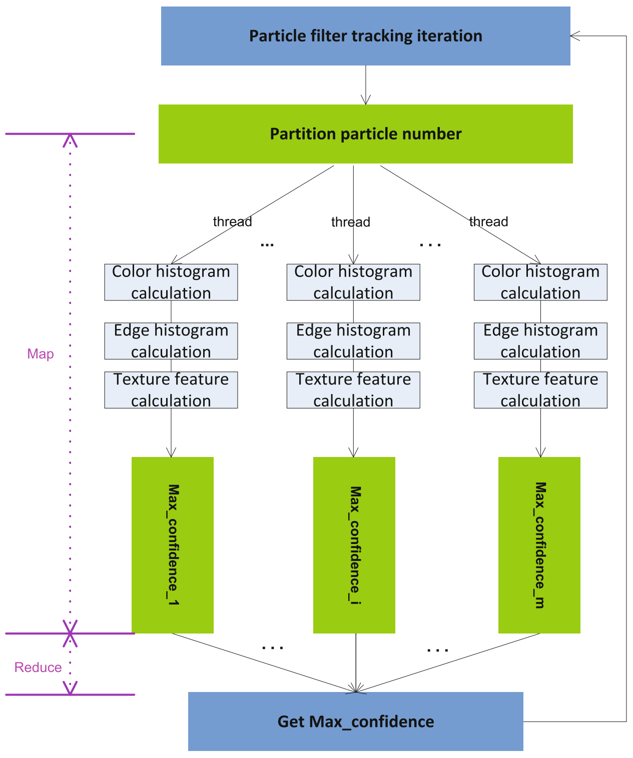

Particle filter tracking iteration
Partition particle number
thread
thread
thread
Color histogram calculation
Edge histogram calculation
Texture feature calculation
Map
Reduce
Max_confidence_1
Max_confidence_i
Max_confidence_m
Get Max_confidence

Fig. 3 Multi-core thread model for particle filter algorithm

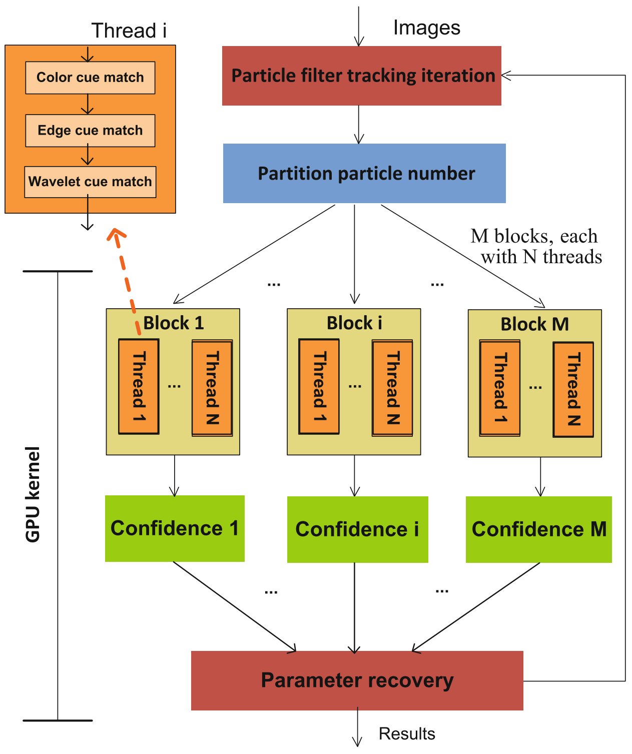


Fig. 4 Parallel particle filter thread model on the GPU

process. For color cue match, the color histogram will be calculated and the Bhattacharya similarity coefficient between reference histogram and calculated histogram. Then the weight will be assigned for each particle according to Bhattacharya similarity coefficient. Finally, the sample weights from all threads are collected and the sample parameter with the maximum confidence is selected to be the final estimation for the target face.

The parallel face-tracking algorithm with particle filter is similar to Algorithm 1 described in Sect. 2.4.2. The only difference is in step 2C. The computation of the weight measurement is performed in a parallel way. All samples are divided into $M$ groups, with each sample calculated by a computing thread.

### 3.4 Parallel particle filter algorithm on GPU and multi-core CPU

To improve the efficiency of particle filter tracking running on the hardware platform, many issues need to be addressed. The main functional modules of the proposed computation architecture are depicted in Fig. 5, and there are three layers: scheduler, MapReduce implementation, and CPU/GPU co-processing.

- *Scheduler* The task scheduler is responsible for paralleling the computation job. Inspired by the Stanford Phoenix (http://csl.stanford.edu/~christos/sw/phoenix/), we have developed a similar scheduler with MapReduce operations. Phoenix is a shared-memory model that could be used for multi-core parallel runtime on the CPU. In the scheduler, the GPU can be viewed as a data-parallel computing device that operates as a co-processor with the main CPU. The CPU takes the management and control role, and the GPU acts as the stream accelerator. The scheduler will schedule the particle filters to CPU or GPU according to the work load and dispatch strategy.

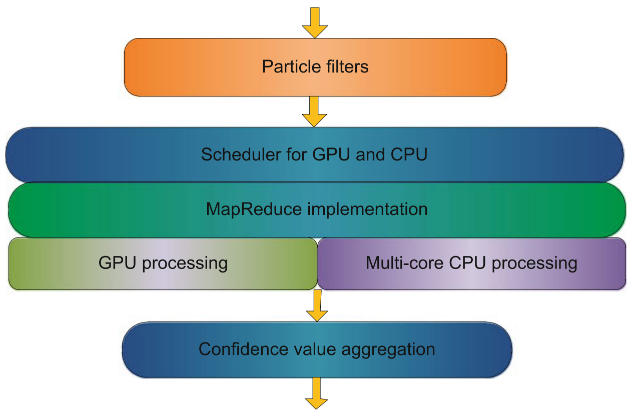

Particle filters
Scheduler for GPU and CPU
MapReduce implementation
GPU processing
Multi-core CPU processing
Confidence value aggregation

**Fig. 5** Co-processing architecture on a heterogeneous system

- *MapReduce* The MapReduce model is an established paradigm for supporting data-intensive computation. The MapReduce implementation provides two primitive operations: (1) a map function to process input key/value pairs and generate intermediate key/value pairs, and (2) a reduce function to merge all intermediate pairs with the same key. With this MapReduce implementation, the computational task will be automatically distributed and executed on multiple machines, multiple processors, and multiple cores.
- *CPU/GPU co-processing* Communication between the CPU and GPU happens in the main memory. Mars [26] is a good co-processing library using CUDA. However, they divided processing of the data statically according to a pre-defined specific ratio, which cannot support complex co-processing schemes.

The co-processing part between the GPU and multi-core CPU is shown in Fig. 6. Here, we take the color histogram calculation as an example. Supposing that there are 2,000 particles for the computation weight, the color histogram of template will be computed by the multi-core CPU. In the case of the task scheduler as mentioned in Fig. 5, the particle dispatch ratio between GPU and multi-core CPU is set up 0.25, which means 400 particle filters sent to the multi-core CPU and 1,600 particle filters sent to the GPU for computing weight, respectively. Once the two parts finish their weight computations, the total confidence sort work will start, and the particle filter with the maximum confidence will be found.

## 4 Experimental implementation and comparison

### 4.1 Tracking accuracy evaluation

A face-tracking system with C++ code has been developed based on Phoenix code and CUDA (http://www.nvidia.com/cuda) on a HP xw8400 workstation. The workstation is configured with dual Intel Xeon 5345 CPUs, totaling eight cores with 4 GB of RAM and a NVIDIA FX4600 card (G80 GPU, 768 MB memory, 12 multiprocessors). A Logitech Pro 4000 web camera was used to capture $320 \times 240$ images for the tracking experiments. Several face-tracking experiments under different illumination conditions are designed to verify the proposed algorithms. As shown in Fig. 7, our tracking algorithm is adaptable to different illuminations and some slight face rotations benefit from online updating strategy, where the facial region is indicated by red rectangles. Face detection is performed every ten frames to verify the tracking results, and a re-initialization process is performed if tracking failure occurs. In our experiments under a sequential mode,

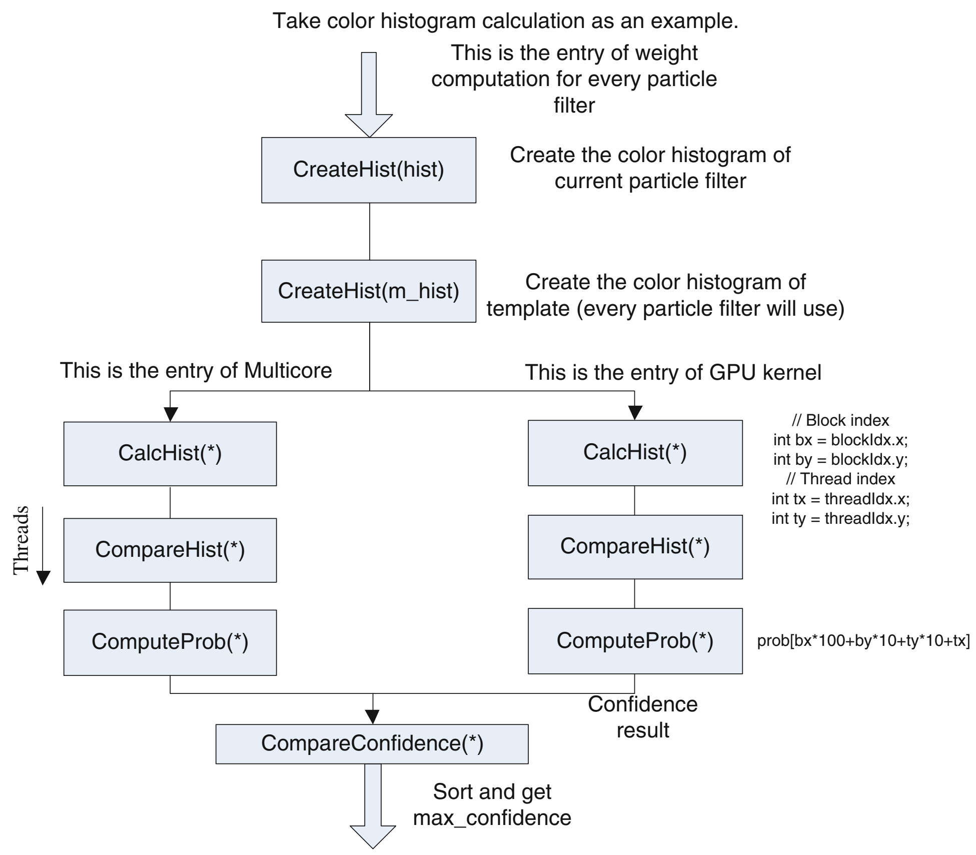

Take color histogram calculation as an example.
This is the entry of weight computation for every particle filter
CreateHist(hist)
Create the color histogram of current particle filter
CreateHist(m_hist)
Create the color histogram of template (every particle filter will use)
This is the entry of Multicore
This is the entry of GPU kernel
CalcHist(*)
CompareHist(*)
ComputeProb(*)
Threads
// Block index
int bx = blockIdx.x;
int by = blockIdx.y;
// Thread index
int tx = threadIdx.x;
int ty = threadIdx.y;
prob[bx*100+by*10+ty*10+tx]
Confidence result
CompareConfidence(*)
Sort and get max_confidence

**Fig. 6** Co-processing between the GPU and multi-core CPU

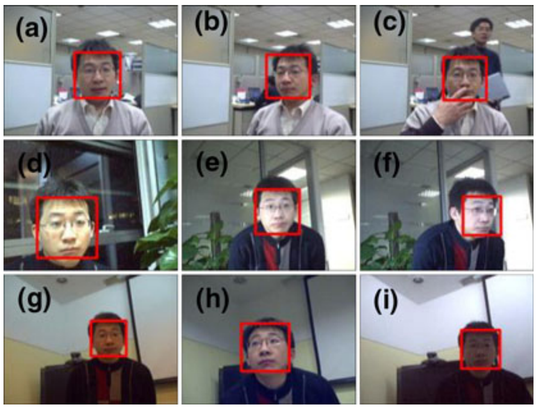


**Fig. 7** Tracking results with different illuminations and views

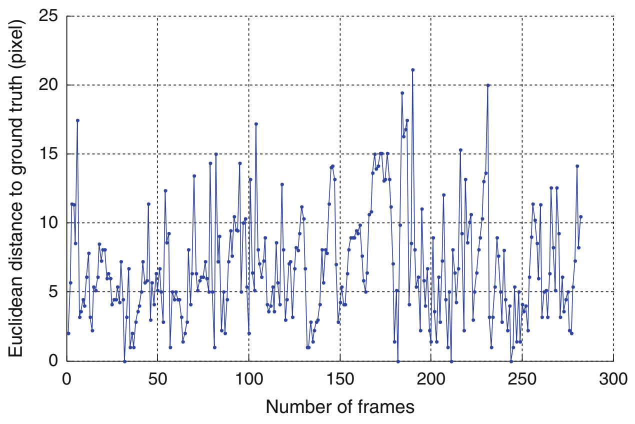


**Fig. 8** Euclidean distance for tracking performance evaluation

the number of particles is set to 1,000 by default. The face number in every frame is 1. The confidence values of color histogram, edge orientation histogram and wavelet feature cues are all set to 1/3. If there are multiple faces in every frame, the numbers of the particle filters should be increased proportionally.

A group of 282 images with face movements was captured to evaluate tracking accuracy performance. Using the manually labeled center positions of face rectangles as the ground truth, the Euclidean distances between the tracking results and the ground truth values are used to demonstrate the accuracy of our face-tracking method, which is shown in Fig. 8. The mean absolute error is 8.43 pixels, and the standard error is 4.06 pixels.

## 4.2 Speedup issue discussions

Normally, the particle filter number is assigned to 100–200 because of sequential computation low efficiency in the real-time application, especially when multiple features are

**Table 1** Three parallel approaches

| No. | Parallel approach | Utility |
|---|---|---|
| 1 | Multi-core | Dual CPUs with 8 cores |
| 2 | GPGPU | NVIDIA G80 FX4600 with 128 stream processors |
| 3 | Multi-core + GPGPU | 8 CPU cores + 128 stream processors |

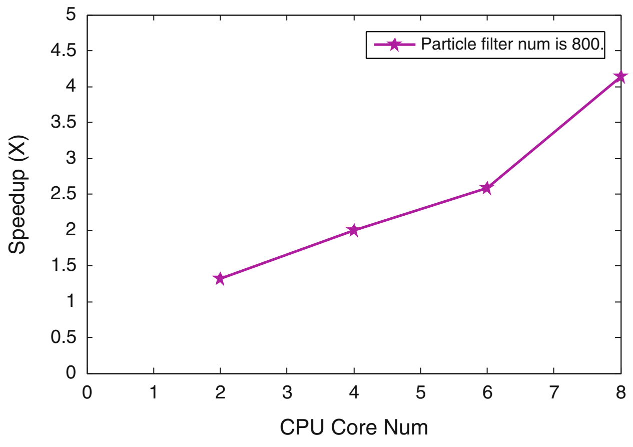


**Fig. 9** Speedup with different computing cores

used. The computing result is from one person (one face) in the investigated frame. If there are several persons in the investigated frame, the computing time will roughly be multiplied by the number of people. The three parallel approaches are listed in Table 1.

We conducted several experiments with sequential and multi-core parallel implementations. The numbers of different particle filters were set at 1,000, 2,000, and up to 10,000, with a step of 1,000. There are eight CPU cores in our workstation, so the number of multi-core that were used range from 1 to 8. To investigate performance with different number cores, we set the particle filter number to 800 and the computing core from 2 to 8. The performance figure is shown in Fig. 9, which shows the maximum speedup (4.2×) obtained when eight cores are used. There is a significant speedup improvement going from six cores to eight cores.

The performance curve based on the multi-core can be seen in Fig. 10. The numbers of multi-core were set to four and eight, and the particle filter number was set from 1,000 to 10,000, respectively. Based on the MapReduce programming model, the particle filter weight computation was assigned to each computing core. The internal MapReduce programming model, improved PThreads library, was used as a thread library. Color histogram, edge histogram, and wavelet histogram calculations were executed in every thread and on every core.

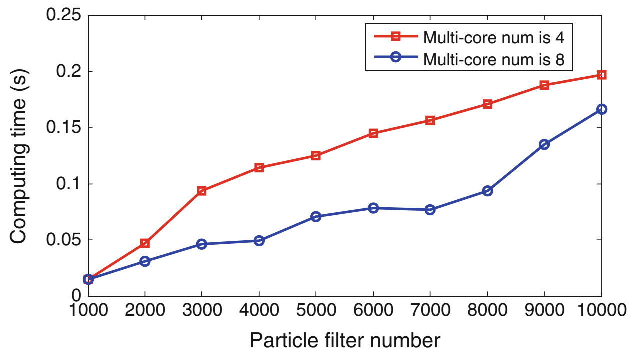


**Fig. 10** The computing time of parallel particle filter on the multi-core

With the particle filter number increased, the computation time in every particle filter weight computation period increased as well. As shown in Fig. 10, it took 0.1 s to finish the particle filter weight computation when the number of particle filters was 3,000, with the multi-core number set to 4. For real-time tracking, a computation time of 0.2 s can be guaranteed when the particle number was set to 10,000. The computation time is derived from the average computing time of a sequence of frames. Based on the results in Fig. 10, the particle filter number can be set to 5,000 or more with the help of the MapReduce thread model and multi-core architecture.

To check the GPU performance more visually, the computation time with different particle filter numbers on the GPU was obtained. The particle filter number was also selected from 1,000 to 10,000, with a step of 1,000. The computing parameters pertaining to color histogram, edge histogram, and texture feature were firstly copied from the main memory to the GPU memory. Then, the device memory for storing results was allocated in advance. The three GPU kernels were used to calculate the color

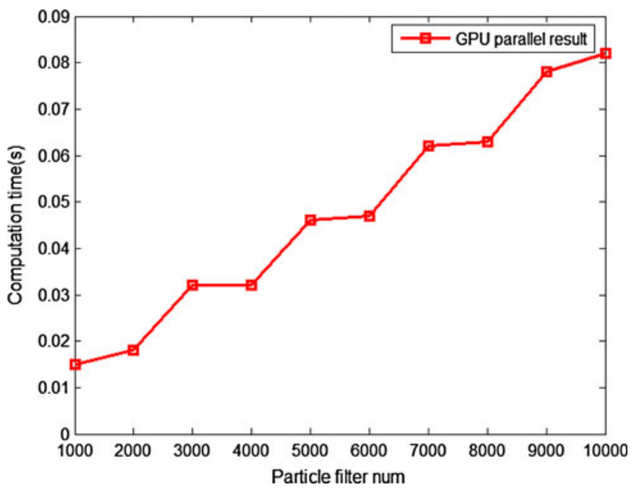


**Fig. 11** The computing time of parallel particle filters on a GPU

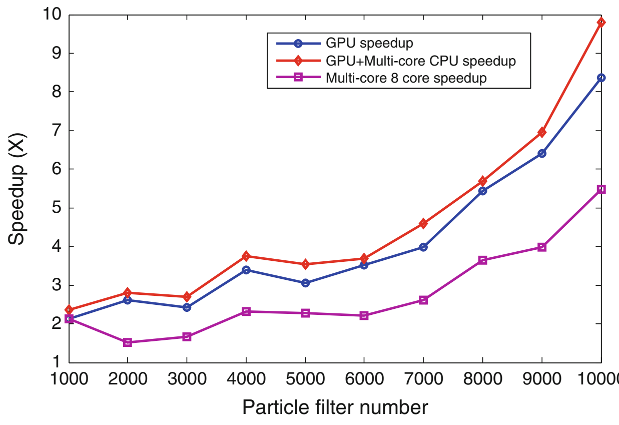


**Fig. 12** Comparison of the speedup effects of different parallel computing methods

histogram kernel, edge histogram kernel, and texture feature kernel. For the parameter *M* and *N* in Fig. 4, we set *M* to 10 as a thread block. Additionally, we set *N* to 100, meaning that there are 100 threads in every block. Once the three kernels were finished, the resulting weight was copied from device memory to host memory. As shown in Fig. 11, the GPU computing time on a weight measure kernel was 0.082 s when the particle filter number was set to 10,000. The computing time is applicable to most case applications.

We would like to make a comparison of the three parallel methods: the multi-core method, the GPGPU method, and the co-processing method. The research experiment includes three parallel scenarios, i.e., task computation on an 8-CPU core, task computation on NVIDA GPU cores, and computation on multi-core workstations with GPGPU. The comparison can be seen in Fig. 12 which contains three speedup comparison curves. The graph shows that a heterogeneous speedup can be better than a GPU speedup and that a GPU speedup is better than multi-core parallel implementation.

## 5 Conclusion

Through the theoretical analysis and the practical face-tracking experiment, we have analyzed the availability of the proposed a particle filter tracking algorithm and processing framework, and the fast face-tracking system can achieve real-time performance using the multi-core and the GPU and can constitute the relative high speedup algorithm for the robust tracking of the face. The experiments of the fast face-tracking based on multi-core and GPU processing which have been implemented, and the experimental results show that the proposed algorithm and processing system can obtain a positive effect and significant speedup performance, and can increase the tracking speed by 8–12 times. At the same time, co-processing between the CPU and the GPU, with the supporting of CUDA, can also increase the ability of the tracking computation more effectively.

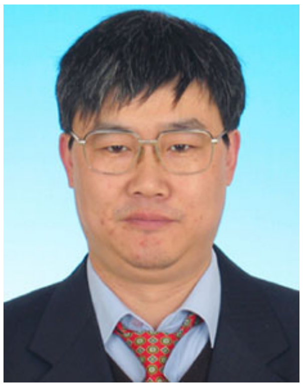
## References

1. Isard, M., Blake, A.: ICONDENSATION: unifying low-level and high-level tracking in a stochastic framework. In: Proceedings of 5th European Conference Computer Vision, vol. 1, (1998)
2. Wu, Y., Huang, T.S.: Color tracking by transductive learning. In: IEEE Conf. on CVPR, vol. 1, pp. 133–138 (2000)
3. Pérez, P., Vermaak, J., Blake, A.: Data fusion for visual tracking with particles. Proc IEEE **92**(3), 495–513 (2004)
4. Vermaak, J., Pérez, P., et al.: Towards improved observation models for visual tracking: selective adaptation. In: Europ. Conf. Computer Vision, pp. 645–660 (2002)
5. Darrell, T., Gordon, G., et al., Integrated person tracking using stereo, color, and pattern detection. In: IEEE Conf. Computer Vision and Pattern Recognition, pp. 601–609 (1998)
6. Jingfeng, L., Yihua, X., et al.: A real-time 3D human body tracking and modeling system. In: IEEE Conf. Image Processing, pp. 2809–2812 (2006)
7. Kalman, R.E.: A new approach to linear filtering and prediction problems. Trans. ASME J. Basic Eng. **82**, 35–45 (1960)
8. Fukunaga, K., Hostetler, L.D.: The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Trans. Inf Theory **21**(1), 32–40 (1975)
9. Arulampalam, S., Maskell, S., et al.: A tutorial on particle filters for on-line nonlinear/non-Gaussian Bayesian tracking. IEEE Trans Signal Process **50**(2), 174–188 (2002)
10. Yilmaz, A., Javed, O., Shah, M.: Object tracking: a survey. ACM J. Comput. Surv. **38**(4), 1–45 (2006)
11. Wang, T., Diao, Q., Zhang, Y., Song, G., Lai, C., Bradski, G.: A dynamic Bayesian network approach to multi-cue based visual tracking. In: Proc. Int. Conf. Pattern Recognition, pp. 167–170 (2004)
12. Wu, Y., Huang, T.S.: A co-inference approach to robust visual tracking. In: IEEE ICCV, vol. 2, pp. 26–33 (2001)
13. Spengler, M., Schiele, B.: Toward robust multi-cue integration for visual tracking. Mach Vis Appl **14**, 50–58 (2003)
14. Brasnett, P., Mihaylova, L., Canagarajah, N., et al.: Particle filtering with multiple cues for object tracking. In: Proceedings of the SPIE, pp. 430–441 (2005)
15. Serby, D., Koller-Meier, E., Van Gool, L.: Probabilistic object tracking using multiple features. ICPR **2**, 184–187 (2004)
16. Zhao, L., Tao, J.: Fast facial feature tracking with multi-cue particle filter. In: Image and Vision Computing, New Zealand, IVCNZ 2007, pp. 7–12, Hamilton, New Zealand, (2007)
17. Triesch, J., von der Malsburg, C.: Self-organized integration of adaptive visual cues for face tracking. In: IEEE Conf. on Automatic Face and Gesture Recognition, pp. 102–107 (2000)
18. Rasmussen, C., Hager, G.D.: Probabilistic data association methods for tracking complex visual objects. PAMI **23**(6), 560–576 (2001)
19. Chen, J., Tiddeman, B.: Multi-cue facial feature detection and tracking. In: International Conference on Image and Signal Processing (ICISP), pp. 356–367 (2008)
20. Liu, H., Yu, Z., Zha, H., et al.: Robust human tracking based on multi-cue integration and mean-shift. Pattern Recognit. Lett. **30**(9), 827–837 (2009)
21. Lozano, O.M., Otsuka, K.: Simultaneous and fast 3D tracking of multiple faces in video by GPU-based stream processing. In:

IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 713–716 (2008)
22. Hendeby, G., Hol, J., Karlsson, R., Gustafsson, F.: A graphics processing unit implementation of the particle filter. In: 15th European Signal Processing Conference (EUSIPCO2007), Poznan, Poland, pp. 1639–1643 (2007)
23. Montemayor, A.S., Pantrigo, J.J., Sanchez, A., et al.: Particle filter on GPUs for real-time tracking. In: Proc. of ACM SIGGRAPH, 2004
24. Happe, M., Lübbers, E., Platzner, M.: A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. J. Real-Time Image Process. doi:10.1007/s11554-011-0212-y (2011) (special issue)
25. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 511–518 (2001)
26. He, B., Fang, W., Govindaraju, N.K., Luo, Q., Wang, T.: Mars: a MapReduce framework on graphics processors. In: Proceedings of PACT 2008 (2008)

## Author Biographies

**Keyan Liu** received the Ph.D. in electrical and computing engineering from Beihang University in 2007. He is now with Nokia Siemens Networks (NSN) to be a senior researcher. From 2007 to 2011, he served for HP labs China as a senior researcher on parallel computing. His current research interests include parallel computing, image/video processing, machine learning.



**Yunhua Li** received his ME degree in mechanical engineering form Jilin University of Technology, Changchun, China, in 1988, and Ph.D. degree in mechatronics from Xi'an Jiaotong University, Xi'an, China, 1994. He was a postdoctoral fellow in BeiHang University and National Chengkung University, from December 1994 to October 1996 and from July 1997 to August 1997, respectively. He has been a Full Professor in Mechanical Engineering in Beihang University since 1999. His research interests include non-linearity dynamics, mechatronic servo control, and power transmission and control of aircraft, mobile robots and vehicle. He has published more than 160 refereed journal and conference papers on the transactions and journals, and 4 books. He has also completed numerous granted research and development projects as the Principal Investigator. Currently, Dr Li is a technical editor of IEEE/ASME Transactions on Mechatronics.