

# Bandwidth-Improved GPU Particle Filter for Visual Tracking

A. S. Montemayor<sup>†1</sup>, J. J. Pantrigo<sup>‡1</sup>, R. Cabido<sup>1</sup> and B. Payne<sup>2</sup>

<sup>1</sup>Escuela Superior de Ciencias Experimentales y Tecnología (ESCET), Universidad Rey Juan Carlos, Madrid, Spain

<sup>2</sup>North Georgia College & State University, Georgia, United States of America

---

## Abstract

*In the literature, many attempts at object visual tracking are performed by particle filtering. This method exploits Monte Carlo techniques to make measurements all over a presegmented image at different positions. Subimage reading at different image locations is a computationally expensive process that can be overcome using a high memory bandwidth hardware. Then, high probability particles, or descriptions of the objects, survive after an evaluation stage and provide new positions of interest that must be evaluated at next time step. In this work, we improve the performance of a particle filter implementation by making use of the higher memory bandwidth rate of a GPU in comparison to a CPU. This improvement is also compared to a previous GPU particle filtering framework with promising results.*

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors; I.4.8 [Image Processing and Computer Vision]: Tracking

---

## 1. Introduction

Efficient object tracking is required by many computer vision application in areas like surveillance and robotics. But tracking is also an important task in several computer graphics environments. Image-Based modeling and rendering (IBMR) systems coexists in the intersection between computer vision and computer graphics [BCD\*03]. In IBMR systems, the geometry and appearance of a computer generated scene is derived from real photographs (or video) to provide an increased degree of realism [Deb96]. Volume generation from edges and silhouettes, articulated motion extraction, featured-based stereo matching, depth maps extraction, structure-from-motion, layered depth images and multi-view scene recovery are typical techniques for IBMR systems [BCD\*03, Sze90, CW93, LS97, DBR00]. Visual tracking methods are related to these techniques as well as to video motion capture systems and automatic animation. Some examples are automatic feature tracking implementations like face or hands tracking applications [BCD\*03].

Recent research in human motion analysis and visual object tracking make use of the Particle Filter (PF) framework.

The PF algorithm enables the modeling of a stochastic process with an arbitrary probability density function, by approximating it numerically with a set of samples called particles [AMGC02]. The main aim of particle filtering is the accurate tracking of a variable of interest as it evolves over time, such as kinematic information of objects in video sequences.

The reasons for the rapid expansion of these IBMR systems are mainly the reduction of the image acquisition hardware prices and the large computational power of current graphics cards. Indeed, this commodity graphics hardware has evolved since the mid-90's giving a considerable amount of programming power to developers in order to customize their rendering effects in real-time. In addition, a consumer graphics processing unit (GPU) has become a kind of inexpensive and programmable stream processor. Many authors have demonstrated that these consumer GPUs boast exceptional peak performance, even superior to the most common and powerful CPUs [THO02, Har03]. They have been used as co-processor for the central processing unit in examples that exploits the power of the GPU for linear algebra calculations [MTP\*04, KW03], physically-based simulations [Har03] and image and volume processing [YW02] among many others [Har02].

In this work, we propose an important improvement on

---

<sup>†</sup> antonio.sanz@urjc.es

<sup>‡</sup> juanjose.pantrigo@urjc.es

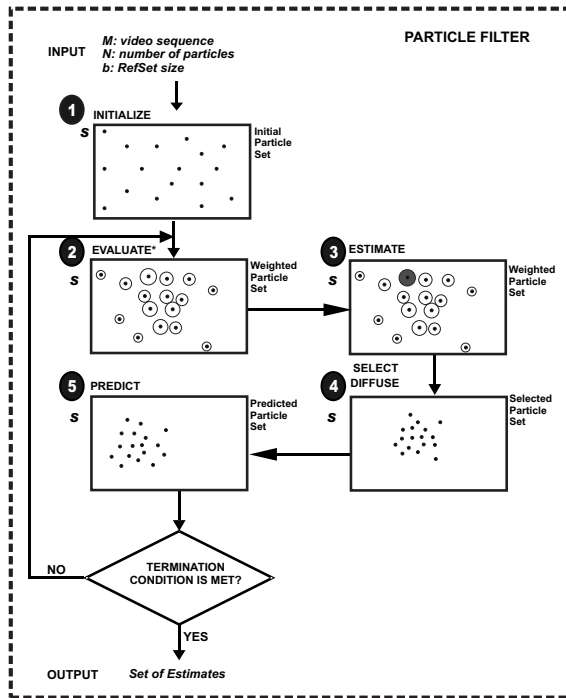


Figure 1: Particle filter algorithmic scheme.

previous CPU/GPU particle filter frameworks [MPSF04, LNB05, MPSF05]. In particular, we have reduced bandwidth requirements in the data allocation stage using GPU texture reads instead of CPU-GPU memory transfers. As a demonstration of the algorithm’s applicability, it has been adapted to real-time 2D tracking in a webcam based system with encouraging results.

## 2. Object Tracking and Particle Filtering

Object tracking is a common problem in many computer vision applications. It deals with state-space variable estimation of interesting features in image sequences and their future prediction. Many probabilistic algorithms have been widely applied which take advantage of knowledge about previous states of the system, thereby reducing the computational cost of an exhaustive search over the whole image. In this framework, the posterior probability density function (pdf) of the state is estimated in two stages: prediction and update. General particle filters are based on discrete representations of probability densities and can be applied to any state-space model [AMGC02]. Discrete particles  $j$  of a set  $(X_t, \Pi_t) = \{(x_t^0, \pi_t^0) \dots (x_t^N, \pi_t^N)\}$  in time step  $t$ , contains information about one possible state of the system  $x_t^j$  and its importance weight  $\pi_t^j$ .

Common implementations of the most simple PF algorithms for 2D visual tracking purposes try to locate objects

of interest using a Monte Carlo approach and extract sub-images from the actual video frame at time  $t$ . Classically, the PF algorithm consists of a sequential set of stages: particle weighting, estimation, selection, diffusion and prediction. Figure 1 describes all these phases.

In a practical approach, particle weight computation is the most expensive stage of the particle filter algorithm, and it has to be executed at each time step for every particle [DBR00]. Basically, this stage scores each particle by means of a likelihood weight between the predicted subimage location and an object template description. The particle filter algorithm itself is computationally expensive but the weight computation is the main bottleneck.

The estimation phase evaluates the best particle by direct weight comparison. The particle with largest weight is considered the best estimator of the position of the object of interest in the actual frame (at time  $t$ ).

Then, those particles with lowest weight are replaced in a probabilistic approach. In this process, particles are randomly chosen proportional to their weights, so particles with high probability can be taken multiple times. This procedure belongs to the selection stage.

As some particles can be chosen several times in the previous step, a diffusion is performed to the survival set. This provides a needed diversification to the set of particles which minimizes repeated evaluations of the same state.

If a motion model of the system is known, we can better predict future estimators. This motion model is applied in the prediction stage. If there is not a *a priori* motion model, we can join the prediction with the diffusion phase in order to search for the trackable object over a restricted, but large enough, region of interest.

### 2.1. Particle Filtering using Graphics Processors

As far as the authors can find, Montemayor et al. [MPSF04] was the first work proposing the use of graphics processors to alleviate the computational cost of a 2D object tracking particle filter system. In that work, the particle weighting stage in terms of subimage comparison was performed on the GPU. It was a preliminary study, but they adapted the sequential image comparison process to a parallel evaluation of many possible locations  $X_t = \{(x_t^0) \dots (x_t^N)\}$  described by each particle  $j$ . Figure 2 summarizes the hybrid CPU/GPU particle filter scheme.

In [LNB05] and [MPSF05], the researchers take advantage of four color channels textures in a SIMD manner to produce more particle evaluations at the same time with almost the same computational cost.

The common approach to the particle weighting stage evaluation on the GPU was to form a texture with subimage portions extracted from the actual video frame at time  $t$ . This data allocation is performed in the CPU with the use

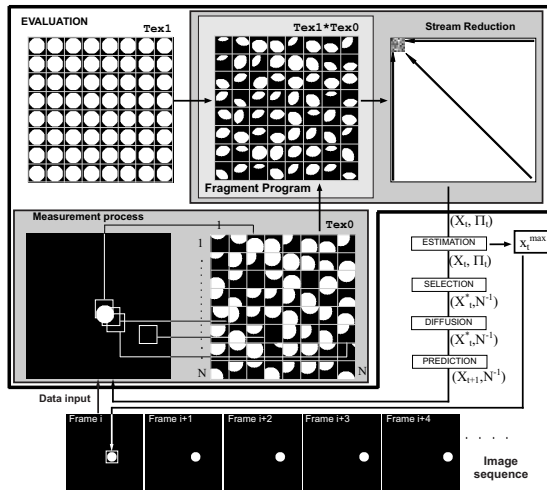


Figure 2: Particle filter scheme with the weighting stage (evaluation) in GPU.

Family	Year	# Trans. ( $\times 10^6$ )	Bandwidth (GB/s)
NV25	2002	63	10.4
NV38	2003	130	30.4
NV40	2004 </td <td>222</td> <td>35.2</td>	222	35.2
G70	2005	302	38.4
R520	2005	321	48

Table 1: Comparison chart: NVXX and G70 from Nvidia, and R520 from ATI (# Trans. is referred to number of transistors of the processor).

of less than optimal memory-copy operations (see measurement process in Figure 2). Once the texture is completely filled, it is compared to the object model texture with the use of a fragment program (see evaluation phase in Figure 2).

### 3. Improving Particle Weighting Stage

In this work, we propose an improvement at the measurement process. Texture creation based on rendering instead of performing expensive memory transfers to and from the CPU would alleviate the bottleneck of data allocation. The underlying hypothesis is that graphics processing units offer higher memory bandwidth ratio than CPUs. This fact is demonstrated comparing some graphics hardware capabilities against common desktop CPUs. Table 1 shows some features of different graphics processors by year of release, including number of transistors and memory bandwidth. For example, an Nvidia GeForce 7800GTX (G70) exhibit memory bandwidth ratios of about 38 GB/s against 10.7 GB/s of an Intel 975X chipset for Pentium 4 (2005).

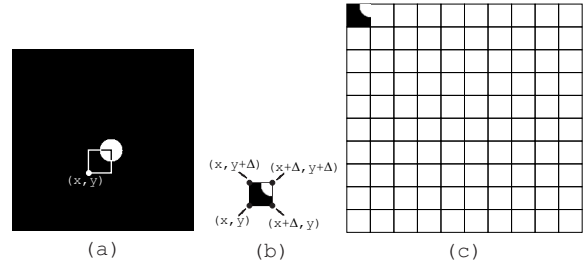


Figure 3: Particle filter scheme with the weighting stage (evaluation) in GPU.

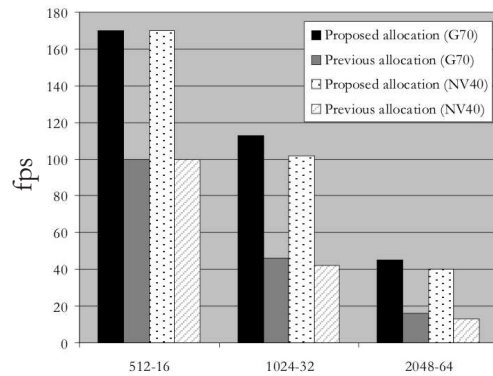


Figure 4: Particle filter scheme with the weighting stage (evaluation) in GPU.

Now, given the video frame at time  $t$  as a texture data in video memory, we propose rendering a collection of textured primitives instead of reordering and transferring data. For this purpose we create a geometry composed of multiple quads completing a larger square region. Then, the measurement process is as follows: given a particle which describes a possible state (Figure 3(a)), we specify texture coordinates  $(s, t)$  for each vertex of the particle corresponding quad (coordinates  $(x, y)$ ,  $(x + \Delta, y)$ ,  $(x, y + \Delta)$  and  $(x + \Delta, y + \Delta)$  in Figure 3(b)). If we need square subimage portions for the object description, it is easy to create a large square texture composed of a collection of small textured quads (Figure 3(c)). The output of a simple rendering can be redirected to a frame buffer object (FBO) that after a single pass over the collection of quads the measurement texture is built. Next, this texture is fed to the hybrid CPU/GPU particle filter system as the normal procedure: performing the evaluation stage, stream reductions in order to get particle weights, data readback to CPU, estimation, selection, diffusion and prediction.

#### 4. Experimental Results

Experiments have been performed using two platforms: a 2.8 GHz Pentium 4 host processor, 512 MB RAM, AGPx8 equipped with an Nvidia GeForce6800 GT (NV40), and a 3.2 GHz Pentium 4, 1GB RAM, PCI-Express x16 with an Nvidia GeForce7800 GTX (G70). Applications were coded in C using OpenGL as rendering API, Cg 1.4 as shading language and Nvidia v81.98 drivers. Input video sequences were in QCIF format (320x240 pixels) and 1024 particles were evaluated at each time step.

Figure 4 shows frame rates of a CPU/GPU object tracking implementation where the number of particles remains constant (1024) but the size of the subimage comparison varies. First and third columns show the proposed data allocation by means of rendering with G70 and NV40 processors while second and fourth columns show previous CPU/GPU execution peak performance for the same GPUs. Groups of columns correspond to subimage windows of 16x16 pixels forming a 512x512 measurement texture, 32x32 pixels and 64x64 respectively.

Experimental results show that the proposed method improves performance of the entire algorithm about 1.7x-3.0x the frame rate of previous memory-copy approaches. We can observe that data reallocation at every time step can be a very important bottleneck when preparing a hardware accelerated solution to a previous CPU application. Here, we alleviate this bottleneck by means of very optimized texture fetching operation, boosting peak performance of the entire algorithm up to 300%.

#### 5. Conclusions

In this work, we present an improvement on previous implementations of hybrid CPU/GPU particle filter algorithm exploiting higher memory bandwidth texture mapping. We propose a customized data allocation by making a texture from small, textured primitives instead of reading back to CPU what was already in video memory. Experimental results have shown a remarkable performance, even frame rates up to 300% compared to previous CPU/GPU particle filtering approaches on the same graphics boards.

#### Acknowledgements

This work has been partially supported by the Spanish Ministry of Education and Science (project TIN2005-08943-C02-02).

#### References

- [AMGC02] ARULAMPALAM S., MASKELL S., GORDON N., CLAPP T.: A Tutorial on Particle Filters for On-line Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Trans. on Signal Processing* 50, 2 (2002), 174–188.
- [BCD\*03] BURSCHKA D., COBZAS D., DODDS Z., HAGER G., JAGERSAND M., YEREX K.: Recent methods for image-based modeling and rendering. *IEEE Virtual Reality 2003 Tutorial #1*, 2003.
- [CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. *Computer Graphics* 27, Annual Conference Series (Aug 1993), 279–288.
- [DBR00] DEUTSCHER J., BLAKE A., REID I.: Articulated body motion capture by annealed particle filtering. In *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)* (2000), pp. 2126–2133.
- [Deb96] DEBEVEC P. E.: *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, 1996.
- [Har02] HARRIS M. J.: GPGPU. General Purpose Computing using GPUs website. <http://www.gpgpu.org>, 2002.
- [Har03] HARRIS M. J.: *Real-Time Cloud Simulation and Rendering*. PhD thesis, University of North Carolina at Chapel Hill, 2003.
- [KW03] KRÜGER J., WESTERMANN R.: Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG)* 22, 3 (Aug 2003), 908–916.
- [LNB05] LANVIN P., NOYER J.-C., BENJELLOUN M.: An hardware architecture for 3d object tracking and motion estimation. In *Proc. of Intl. Conf. on Multimedia and Expo (ICME)* (2005).
- [LS97] LENGYEL J., SNYDER J.: Rendering with coherent layers. *Computer Graphics* 27, Annual Conference Series (Aug 1997), 233–242.
- [MPSF04] MONTEMAYOR A., PANTRIGO J., SÁNCHEZ A., FERNÁNDEZ F.: Particle filter on GPUs for real-time tracking. In *Proc. of ACM SIGGRAPH 2004* (2004).
- [MPSF05] MONTEMAYOR A., PANTRIGO J., SÁNCHEZ A., FERNÁNDEZ F.: Particle filter on GPUs for multiple object tracking in HCI applications. In *Proc. of ACM SIGGRAPH 2005* (2005).
- [MTP\*04] MCCOOL M., TOIT S. D., POPA T., CHAN B., MOULE K.: Shader algebra. *ACM TOG* 23, 3 (2004), 787–795.
- [Sze90] SZELISKI R.: *Real-Time Octree Generation from Rotating Objects*. Tech. rep., Digital Equipment Corporation, Cambridge Research Lab, Dec 1990.
- [THO02] THOMPSON C. J., HAHN S., OSKIN M.: Using modern graphics architectures for general-purpose computing: A framework and analysis. In *Int. Symposium on Microarchitecture (MICRO)* (2002).
- [YW02] YANG R., WELCH G.: Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools* 7, 4 (2002), 91–100.