# Particle Filter Speed Up Using a GPU

**John Sacha, Andrew Shaffer**
**Applied Research Laboratory**
**The Pennsylvania State University**
**jrs9@psu.edu**
**(814) 863-4162**

## Introduction

For many years, Kalman filters and related formulations have served as the standard basis for object tracking applications. Such techniques are computationally efficient (an important consideration when processing resources are limited), and provably optimal when their underlying assumptions of Gaussianity, linearity, and stationarity are satisfied. However, these requirements are often violated in practical applications. Sequential Monte Carlo techniques, so-called *particle filters*, represent probability densities as collections of weighted point-masses, and the fundamental density propagation integral equations are solved via Monte Carlo techniques [1]. Manipulating density functions represented in this manner is computationally expensive, but the introduction of cheap computational power in the form of multi-core architectures offers the promise of using of such methods for real-time embedded applications.

Particle filters can facilitate the exploitation of *a priori* information in the form of (non-linear) hard-truncated position, speed, and acceleration constraints, and the placement of environmental boundaries. Examples of applications include multi-modal measurements, passive source localization [2,3], and tracking spatially-extended objects [4].

## Particle Filter Algorithm

The particle cloud approach is both intuitively appealing and computationally elegant. The generic algorithm is as follows.

## Algorithm I: Particle Filtering

Step 1: **Initialization**

- Create a collection of $N$ particles: $S = \left\{ \mathbf{x}^{[i]}, w^{[i]} \mid i = 1, \ldots, N \right\}$, where $\mathbf{x}^{[i]}$ is a state vector and $w^{[i]}$ is a scalar weight.

LOOP (over time):

Step 2: **Prediction**

- Project particles forward according to the transition equation: $\mathbf{x}^{[i]} \leftarrow \mathrm{f}\left( \mathbf{x}^{[i]}, \mathbf{v}^{[i]} \right)$, where $\mathbf{v}^{[i]}$ is an instance of the process noise.

Step 3: **Apply Constraints**

- Apply physical constraints: Particle states that violate *a priori* conditions such as forbidden regions or maximum velocity limits are eliminated: $w^{[i]} \leftarrow 0$.

Step 4: **Apply Measurement**

- Adjust particle weights based on measurement at time $k$ via $w^{[i]} \leftarrow w^{[i]} \mathrm{f}\left( \mathbf{x}^{[i]} \mid \mathbf{z}_k \right)$

Step 5: **Resample**

- Perform a random sampling (with replacement),when the effective number of particles becomes small: $S \leftarrow \mathrm{r}(S)$

Step 6: **Calculate application-specific ensemble statistics.**

END;

Each particle has an associated weight. This use of weights allows a high degree of computational flexibility. It makes it simple to apply the constraints and measurements to the sample set, and more importantly, it can simplify generating samples from a desired density $p_X(X)$: i.e., generate particles according to a *proposal distribution* $q_X(X)$ (for which it is easy to generate samples, such as uniform or Gaussian) and weigh the samples according to $p_X(X)/q_X(X)$.

As the iteration progresses, information tends to become concentrated in a small subset of the particles. (This is true even if weights do not go totally to zero.) This is computationally wasteful as well as leading to loss of accuracy. This is the reason for the resampling step. When the effective number of particles, given by

$$N_{eff} = \left[ \sum_{i=1}^{N} w_i \right]^2 \Big/ \sum_{i=1}^{N} w_i^2$$

falls below some pre-specified fraction of the desired population, *N*, it is advantageous to eliminate low-weight particles. In the resampling step , *N* particles are chosen from the existing population (with replacement), with the probability of selection being proportional to the particle weight. Variation is reintroduced in the subsequent prediction step through the injection of noise according to the transition model.

## Computational Complexity

The ability of the particle filter to accommodate arbitrary probability densities comes at the price of increased computational complexity. For a simple planar tracking problem with four-dimensional state $(x,y,v_x,v_y)$, a particle filter with $N$=1000 particles can be about two orders of magnitude more computationally expensive than a Kalman filter.

Fortunately, the nature of the algorithm makes it a good candidate for parallel processing, and implementations have been produced for a variety of architectures. The prediction step and weight adjustment processes are "embarrassingly parallel" computations that do not require communication between different particles. However, the resampling step depends upon the entire ensemble of particles: the standard approach performs a cumulative summing of the weights, the generation of sorted random numbers and a sorted list merge. Resampling is regarded as the major challenge in producing a parallel formulation of particle filtering [5-8].

Here we examine the implementation of particle filter components on a multicore GPU, using the CUDA extensions to the C/C++ language. Key issues include parallel prefix summation, and tradeoffs between using an $O(N)$ linear search versus an $O(N \log N)$ (but readily parallelizable) binary search. Order-of-magnitude speedups over a conventional general-purpose CPU were observed for some components.

## Acknowledgements

This work was sponsored by the Office of Naval Research under contract number N00014-05-G-0106/0006. Opinions, interpretations, conclusions and recommendations are those of the authors and not necessarily endorsed by the United State Government.

## References

[1] Ristic, B.; Arulampalam, S.; Gordon, N., *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Boston, Artech House, 2004.

[2] Baggeroer, A.B.; Kuperman, W.A.; Mikhalevsky, P.N.; "An overview of matched field methods in ocean acoustics," *IEEE Journal of Oceanic Engineering*, Vol. 18, No. 4, Oct. 1993, pp. 401–424

**[3]** Kraut, S;, Krolik, J.; "Moving Target Depth Estimation For Passive Sonar, Using Sequential Resampling Techniques," *Proc. Adaptive Sensor Array Processing Workshop***,** 14 March 2001, (reprint from Dept. of ECE, Box 90291 Duke University Durham, NC 27708-0291**).**

[4] Vermaak, J.; Ikoma, N.; Godsill, S.J.; "Sequential Monte Carlo framework for extended object tracking," *IEE Proceedings Radar, Sonar and Navigation*, Vol. 152, No. 5, October 2005, pp. 353–363.

[5] Bolić, M., Djurić, P. M., Hong, S., "Resampling Algorithms and Architectures for Distributed Particle Filters," *IEEE Trans. Signal Processing*, Vol. 53, No. 7, July 2005, pp. 2442-2450.

[6] Hong, S., Djurić, P. M., "High Throughput Scalable Parallel Resampling Mechanism for Effective Redistribution of Particles," *IEEE Trans. Signal Processing*, Vol. 54, No. 3, March 2006, pp. 1144-1155.

[7] Míguez, J., "Analysis of parallelizable resampling algorithms for particle filtering," *Signal Processing*, Vol. 87, No. 12, December 2007, Pages 3155-3174.

[8] Lozano, O. M., Otsuka, K., "Simultaneous and fast 3D tracking of multiple faces in video by GPU-based stream processing," *Proc. 2008 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 31 March 2008- 4April 2008, pp. 713 – 716.