

Improving GPU Particle Filter with Shader Model 3.0 for Visual Tracking

Antonio S. Montemayor*
URJC (Madrid, SPAIN)

Juan José Pantrigo
URJC (Madrid, SPAIN)

Raúl Cabido
URJC (Madrid, SPAIN)

Bryson R. Payne
North Georgia College & State University (USA)

Ángel Sánchez
URJC (Madrid, SPAIN)

Felipe Fernández
FI - UPM (Madrid, SPAIN)

1 Introduction

Human-Computer Interaction is evolving towards non-contact devices using perceptual user interfaces. 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.

The DirectX Shader Model is a common framework for accessing graphics hardware features in terms of shading functionality. In particular, Shader Model 3.0 compliant graphics cards must support features such as dynamic branching, longer shader programs and texture lookups from vertex buffers, among others.

In this work, we propose new improvements on previous CPU/GPU Particle Filter frameworks [Montemayor et al. 2004; Lanvin et al. 2005]. In particular, we have reduced bandwidth requirements in the data allocation stage using GPU texture reads instead of CPU-GPU memory transfers. But more importantly, using new features in Shader Model 3.0 we can move all the previous particle filtering CPU stages to the GPU, keeping all the computation on the video card and avoiding expensive data readback.

2 Method Overview

Common implementations of PF algorithms for 2D visual tracking purposes try to locate objects of interest using a Monte Carlo approach and extract subimages 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.

Previous hardware accelerated PF implementations computed just the particle weighting stage on the GPU. This stage can be computationally expensive, and the GPU was doing a good job. However, this approach had two main bottlenecks: data allocation and readback from video memory to host memory. Previously, every considered subimage of the actual video frame was organized as a grid pattern in CPU, and downloaded to GPU memory as a texture at every time step. 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, which improves performance about 1.7x-2.0x the frame rate.

Once data was stored in GPU memory, a fragment program was enabled and a stream-reduction was performed to extract the weighting factor π_t^j corresponding to each particle j (possible coordinates x and y of the trackable object) at time t . Then, subsequent stages (estimation, selection, diffusion and prediction) were executed on the CPU. Now, we propose an entirely GPU solution for the rest of the PF stages that is summarized in Figure 1. The weighting texture can be used to obtain the best description of the state in the estimation phase, and to replace those particles with very low weight in the selection phase. Then, we use a texture filled with random values

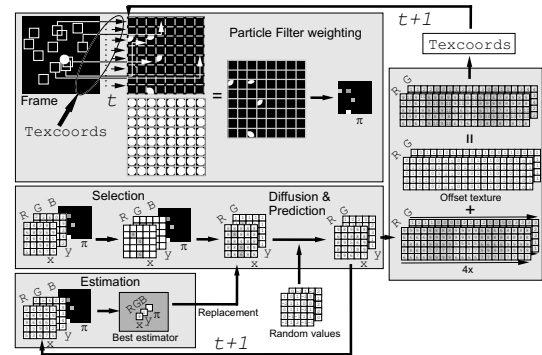


Figure 1: GPU Particle Filter scheme.

to diversify the set of particles and to get the *a priori* estimators for the next time step $t + 1$. The key point is the feedback loop to geometry by fetching texcoord values from a texture. We can get new coordinates for the measurement stage at time $t + 1$ by enlarging the final texture result, providing four texture coordinates for each vertex primitive per initial quad. However, before reading fragments as texture coordinates we have to add an offset to the three new texcoords which expresses the relative position (x,y) of each corner from the original. Features in Shader Model 3.0 tested on an Nvidia GeForce 7800GTX GPU exhibit frame rates up to 250%-325% of that of previous CPU/GPU particle filtering algorithms on the same graphics board (texture sizes from 512x512 to 1024x1024).

3 Conclusion

In this work, we present an idea to obtain an entirely GPU accelerated particle filter system by exploring the newest graphics hardware capabilities. Taking advantage of a higher memory bandwidth ratio, 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. In addition, we close the PF execution loop on the GPU using new FBO-PBO-VBO functionality with very successful results. This is now possible through Shader Model 3.0's vertex texture capabilities.

References

LANVIN, P., NOYER, J.-C., AND BENJELLOUN, M. 2005. An hardware architecture for 3d object tracking and motion estimation. In *Proc. of Intl. Conf. on Multimedia and Expo (ICME)*.

MONTEMAYOR, A., PANTRIGO, J., SÁNCHEZ, A., AND FERNÁNDEZ, F. 2004. Particle filter on GPUs for real-time tracking. In *Proc. of ACM SIGGRAPH 2004*.

*e-mail: antonio.sanz@urjc.es

⁰This research has been supported by CICYT TIN2005-08943-C02-02