

Scalable Particle Filter Framework for Visual Tracking

Raúl Cabido, Antonio S. Montemayor*, Juan José Pantrigo
Universidad Rey Juan Carlos (Madrid, SPAIN)

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

1 Introduction

In this paper, we present a work-in-progress toward multiple object tracking exploiting the GPU as the main processor. This work is based on new Shader Model 3.0 capabilities and recent research on GPU tracking, such as [Montemayor et al. 2006], extending the previous scope to a scalable and reusable framework.

The GPU, as a data-parallel processor architecture, needs an independent-data (parallelizable) processing model in order to provide the best performance. One of the most popular methods for visual tracking is the Particle Filter (PF) algorithm. The PF algorithm enables the modeling of a stochastic process with an arbitrary probability density function by approximating it numerically with a weighted set of samples called particles. Those particles are independent one from each other, and thus, PF becomes ideal for the GPU computation model.

In [Montemayor et al. 2006], the authors proposed improvements on previous CPU/GPU Particle Filter frameworks [Lanvin et al. 2005]. They reduced bandwidth requirements in the data allocation stage and used new Shader Model 3.0 features, moving all the previous particle filtering CPU stages to the GPU. That work is summarized in Figure 1.a, where many particles are used to maintain information about the actual position of one object of interest. In this case, each particle, p_i , is composed of two variables, an estimated position of interest and the weight of that estimation, $p_i=(x_i, \pi_i)$. As the algorithm works with a considerable number of particles, some of the particles will approximate the actual position of the trackable object, so replacing those that do not match with the best estimators leads to a collection improvement.

When multiple targets are considered, most common particle filter approaches assign the weight of the estimation to the likelihood of the multiple estimations together, $p_i=(x_{ij}, \sum_j \pi_j)$. This means that there is one global weight for many estimated positions. As of this writing, the current stage of the present work fits the example summarized in Figure 1.b, with successful multiple object tracking at about 300 frames per second in 320x240 video resolutions (theoretically, if image acquisition were possible at that rate) for objects up to 16x16 pixels in size.

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 . Those subimages are the measurements of the algorithm, and their corresponding positions of extraction are guided by the stochastic process of the particle filtering. As found in the literature, the PF algorithm consists of a sequential set of stages: particle weighting, estimation, selection, diffusion and prediction.

The weighting stage for a 2D tracking problem requires a direct comparison between image-region measurements and the object of interest. After an object-preprocessing stage (done by object detection, skin segmentation, etc.), measurements are extracted by con-

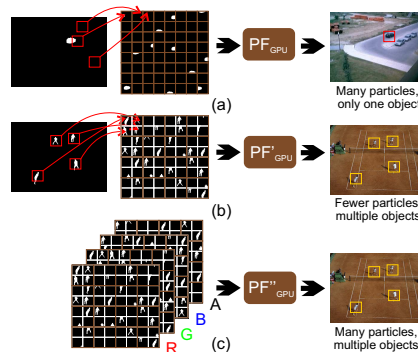


Figure 1: Different Particle Filter schemes.

sidering positions found in the particle set. To this end, a fragment shader can be used to preprocess the actual video frame at time step t . Then, a collection of textured quads with the coordinates given by the particle estimation, as suggested in the first stage of Figures 1.a and 1.b, is rendered to a frame buffer object (FBO). The strategy considered, between one target or multiple targets, differs in the fact that particle weights are responsible for one or many subimages.

Differences in the data layout of the second approximation make the rest of the PF stages more difficult, taking into account the limitations of the GPU computation model. In particular, the particle set is stored in a texture (*particle texture*), using its RGB channels to store information about positions of interest and estimation weights (coordinate x in the red channel, y in the green and the weight π in the blue). Particles are updated with the feedback from the weighting stage, that is, the texture evaluation measurement. However, this required level of abstraction on the GPU creates a correspondence problem between each particle and its corresponding measurement.

At this point, many configurations are available, taking into account the number of particles, the number of trackable objects and their sizes, etc. It is very important to emphasize that, with the capabilities in Shader Model 3.0, it is possible to render the resulting modifications on the *particle texture* to the vertex coordinates of future measurements. From here, it is possible to build a particle filter model where different configurations can be easily evaluated on GPU. For optimization purposes, this framework has to use the RGBA channels of the measurement texture as illustrated in Figure 1.c. This is easily accomplished as an extension of the work presented herein.

3 Conclusion

In this work, we present the basis for a scalable framework for visual tracking with the successful demonstration of a multiple object tracking particle filter that runs entirely on the GPU. Through Shader Model 3.0's vertex texture capabilities and FBO-PBO-VBO functionality, it is possible to break up many correspondence problems among different types of textures. This abstraction makes it possible to achieve visual tracking from particle filters completely on the GPU.

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

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

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., CABIDO, R., PAYNE, B., SÁNCHEZ, A., AND FERNÁNDEZ, F. 2006. Improving GPU particle filter with shader model 3.0 for visual tracking. In *Proc. of ACM SIGGRAPH 2006-Research Posters*.