An hardware architecture for 3D object tracking and motion estimation

P. Lanvin   J.-C. Noyer   M. Benjelloun

Laboratoire d'Analyse des Systèmes du Littoral
Université du Littoral Côte d'Opale
50 Rue Ferdinand Buisson, B.P. 699, 62228 Calais Cedex, France

Abstract

We present a method to track and estimate the motion of a 3D object with a monocular image sequence. The problem is based on the state equations and is solved by a sequential Monte Carlo Method. The method uses a CAD model of the object whose projection can be compared directly with the pixels of the image. The advantage is to obtain a better accuracy and a direct estimation of the pose and motion in the 3D world.

However, this algorithm needs a massive load in computing. For real-time use, we develop in this paper a distributed algorithm that dispatches the processing between the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) of a consumer-market computer. Some experimental results show that it is possible to obtain an accurate 3D tracking of the object with low computing costs.

1. Introduction

The aim of this paper is to describe the implementation of a 3D object tracking and motion estimation algorithm from an intensity image sequence. This problem has been widely studied and has led to many methods which can be classified in two categories. In the feature-based methods [3], 3D motion is determined by tracking of 2D primitives (points, lines, regions, ...) in the image sequence. The model-based approaches are more robust because they use 3D object informations [2, 4]. Moreover, the 3D motion is described by physics equations and is easier to model than the projected 2D motion obtained from image sequence analysis. Finally, the knowledge of object geometry allows an easy management of auto-occlusions.

The proposed algorithm directly uses the grey level pixels without any preprocessing which disrupts and weakens the estimation. The problem lies in a state modelling [5]: the dynamics equation describes the evolution of the object and the measurement equation links the grey level pixels with the state vector (which is composed of the pose and motion parameters of the object). This modelling shows obviously strong non-linearities. We use the sequential Monte Carlo methods [6] (known as particle filtering, bootstrap filtering, condensation algorithm, ...) to solve this non-linear estimation problem. This kind of methods can deal with non-linear models and non-Gaussian statistics. They are only limited by the finite number of particles.

A common problem of sequential Monte Carlo Methods is their significant overload in terms of computing cost, though the computer evolution makes them more and more affordable. One proposes here a new implementation of the algorithm on a standard PC architecture which is composed by a central processing unit (CPU) and a graphics processing unit (GPU). Compared to previous works [2, 1], this new approach needs a full restatement of the particle-based solution, which leads to significant speed-ups and better estimation accuracies. Modern GPUs are able of complex 3D rendering and more general computer vision applications [7, 8]. They are theoretically more powerful than CPU, with a high price/performance ratio. This architecture allows to fully use the processors and exploits a form of parallelism between CPU and GPU. The global performance of the architecture is thus increased, or at least allows to free up CPU resources which can be used for other tasks.

The document is structured as follows: the problem is first modelled with the state formalism. The particle filtering is then detailed and the next paragraph describes the proposed architecture for the implementation and several improvements. The last part allows to evaluate the performances of the algorithm/architecture with two image sequences.

2 3D state modelling

State modelling : We propose a state modelling of this 3D pose and motion estimation problem [1]. The equations describe the 3D evolution of the object (dynamics equation) and partial observation (measurement equation) by the sensor.

The object is characterized by pose parameters defined in its local reference frame (Fig. 1) and by motion parameters. The object is described by a textured mesh obtained by a

rendering software for instance.

$$X_t = [x_t, y_t, z_t, \theta_t^x, \theta_t^y, \theta_t^z, T_t^x, T_t^y, T_t^z, V_t^{\theta^x}, V_t^{\theta^y}, V_t^{\theta^z}]^T \tag{1}$$

x_t^G, y_t^G, z_t^G are the 3D coordinates of the center of gravity, $\alpha_t^x, \alpha_t^y, \alpha_t^z$ are Euler angles of the 3D object, T_t^x, T_t^y, T_t^z are the coordinates of the translation vector, $V_t^{\alpha^x}, V_t^{\alpha^y}, V_t^{\alpha^z}$ are the angular speeds.
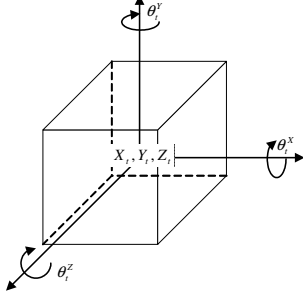

Figure 1: Pose of the object

Dynamics equation : the object evolves in a rigid translation/rotation motion with the following equation:

$$X_t = F X_{t-1} + W_t \tag{2}$$

F is the system's flow which characterizes the evolution of the object at each time instant:

$$F = \begin{pmatrix} I_{6\times6} & I_{6\times6} \\ 0_{6\times6} & I_{6\times6} \end{pmatrix} \tag{3}$$

W_t is a dynamics noise with zero mean and Q_t covariance matrix.

Measurement equation : the object is viewed by a CCD camera which delivers an $N_x \times N_y$ image sequence of the scene:

$$Z_t = H(X_t) + V_t \tag{4}$$

H is the function which links the object model (appearance and position) with the pixels of intensity image, Z_t is the intensity image provides by the sensor at time t and V_t are independent additive Gaussian noises with μ_t mean and R_t covariance matrix.
To sum up, the state modelling can be written as follows:

$$\begin{cases} X_t &= F X_{t-1} + W_t \\ Z_t &= H(X_t) + V_t \end{cases} \tag{5}$$

3   Particle solution of this problem

The measurement equation shows obviously that the state model is strongly non-linear and needs suitable methods to solve this problem. The proposed solution relies on the particle filtering. The probability density function, solution of the filtering problem, is approximated by N random particles whose supports X_t^i and weights p_t^i are conditioned by the measurement Z_t. The particle solution can be split in five parts:

1. **Initialisation :** each particle X_0^i is initialized with the *a priori* law $P(X_0)$. The weights p_0^i are set to $1/N$;

2. **Evolution :** the particles evolve in state space according to the dynamics equation (Eq. 2), via N randoms trials W_t^i with law $P(W_t)$ (zero mean, Q_t covariance matrix) ;

3. **Weighting :** this step adjusts the weights of the particles with the new available measurement and the object projection for the particle i :

$$p_t^i = \frac{P(Z_t|X_t^i)}{\sum\limits_{j=1}^{N} P(Z_t|X_t^j)} p_{t-1}^i \tag{6}$$

4. **Estimation :** the filter delivers the estimation as the weighted sum of the particles:

$$\hat{X}_{t|t} = \sum_{i=1}^{N} p_t^i X_t^i \tag{7}$$

5. **Redistribution :** it is known that the basic particle procedure does not prevent some of the weights to be low compared to those of others particles and therefore to poorly contribute to the performance of the estimator. The redistribution step consists to reallocate particles with high probability.

4. Hardware Implementation

4.1. Presentation

The architecture used to implement the algorithm is a consumer-market PC. It is composed of three functional sets (Fig. 2 left): a central processing unit (CPU) which organizes the several steps of the particle algorithm and a graphics processing unit which performs the particles image reconstruction. CPU and GPU communication is made through a bidirectionnal data bus. The GPU takes care of the rendering tasks obviously better than the CPU, while the CPU can concentrate himself on the particle filtering tasks. As a result, the parallelism between CPU and GPU increases the overall performance of the algorithm.

4.2. Particle weighting

The weighting procedure uses the measurement at time t and the partial reconstructed measurement for the particle. This reconstruction is made by the GPU which uses the object parameters and the object model to project the 3D object model in four steps :
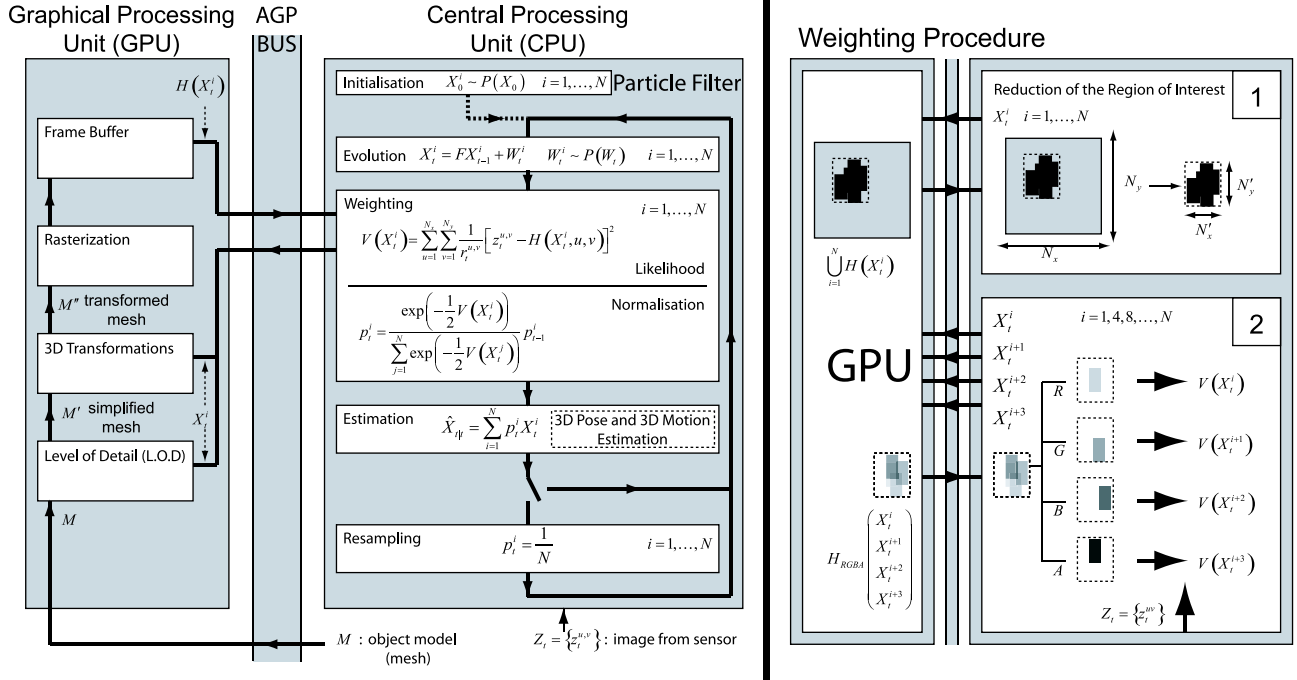
Figure

Graphical Processing Unit (GPU) | AGP BUS | Central Processing Unit (CPU) | Weighting Procedure

$H(X_t^i)$

Frame Buffer

Rasterization

M'' transformed mesh

3D Transformations

M' simplified mesh X_t^i

Level of Detail (L.O.D)

M

Particle Filter

Initialisation $X_0^i \sim P(X_0)$ $i = 1,...,N$

Evolution $X_t^i = F X_{t-1}^i + W_t^i$ $W_t^i \sim P(W_t)$ $i = 1,...,N$

Weighting $i = 1,...,N$
$$V(X_t^i) = \sum_{u=1}^{N_x} \sum_{v=1}^{N_y} \frac{1}{r_t^{u,v}} \left[z_t^{u,v} - H(X_t^i, u, v) \right]^2$$
Likelihood

Normalisation
$$p_t^i = \frac{\exp\left(-\frac{1}{2}V(X_t^i)\right)}{\sum_{j=1}^{N} \exp\left(-\frac{1}{2}V(X_t^j)\right)} p_{t-1}^i$$

Estimation $\hat{X}_{t|t} = \sum_{i=1}^{N} p_t^i X_t^i$ 3D Pose and 3D Motion Estimation

Resampling $p_t^i = \frac{1}{N}$ $i = 1,...,N$

M : object model (mesh)

$Z_t = \{z_t^{u,v}\}$: image from sensor

Reduction of the Region of Interest   1
X_t^i $i = 1,...,N$
$N_y \rightarrow N_y'$ N_x' N_x

$\bigcup_{i=1}^{N} H(X_t^i)$

GPU

$H_{RGBA} \begin{pmatrix} X_t^i \\ X_t^{i+1} \\ X_t^{i+2} \\ X_t^{i+3} \end{pmatrix}$

X_t^i $i = 1, 4, 8, ..., N$ 2
X_t^{i+1}
X_t^{i+2}
X_t^{i+3}

R → $V(X_t^i)$
G → $V(X_t^{i+1})$
B → $V(X_t^{i+2})$
A → $V(X_t^{i+3})$

$Z_t = \{z_t^{uv}\}$

Figure 2: **Left :** Processing architecture. **Right :** Weighting procedure synopsis.

1. **Level of detail :** the object model is first simplified according to the camera/object distance. This step can improve the speed of the transformation step, without any really lost of precision ;

2. **Transformation :** the object geometry is transformed from his local reference frame to the camera reference frame. Then it is projected in the image plane with the sensors parameters ;

3. **Rasterization :** the object is rendered by discretization of each triangle which composes the mesh. The textures and colors are applied here.

4. **Frame Buffer transfer:** next, the image result is transfered to system memory for the likelihood computation.

In spite of the joint use of CPU and GPU, the great number of particles and the size of images do not allow to track the object in low computational time. Two improvements (schematically presented on Fig.2 : right) have been studied in order to limit this drawback.

The first improvement eliminates the redundancy in computation, which happens in the weighting stage: some areas of the image do not give informations in the likelihood computation because they are identical for every particle. Consequently, one can restrict the processing to a region of interest (ROI). The determination of this ROI is made in two steps. First, the set of particles' object models is rendered in the same image, which is then transfered in the system memory. Next, the bounding-box of the ROI is determined by the CPU and is used to restrict the amount of information to be transfered from the GPU. The images used to render the objects are cropped to the ROI and their new dimensions allow to restrict the data to be transfered from the GPU.

The second improvement restricts the GPU-to-CPU transfers and speeds up the processing. The images provided by the sensor and the reconstructed images for each particles are 8 bits grey levels (one channel per pixel). The most easier approach consists of working with luminance images in the GPU, but the drawback is that the modern GPUs are optimized for RGBA color images (four channels per pixel). With a RGBA image, the four channels in one pixel can be processed using SIMD instructions which means that the instructions can be applied simultaneously on the four data. To fully exploit this functionnality, we combine 4 reconstructed images into one RGBA color image returned by the GPU. To this end, the particle-based reconstructions of the object are projected sequentially while only allowing the rendering on a single component of the image (R for particle i, G for particle i+1, ...). The GPU has a channel mask which allows to prevent writting operations in one or more channels. Then, the CPU get the corresponding RGBA image and computes the likelihood for 4 particles simultaneously.

3

5. Results

The algorithm is applied on synthetic image sequence and real image sequence to evaluate the performances. In the first sequence, a cube evolves in a rigid translation/rotation motion. In the second sequence, a polyhedral object evolves in a rotation motion (approximately 8 degree per image). The test hardware is a PC composed of a P4 type processor at 3GHz (FSB800, Bus AGP x8). The graphics part is performed by a 6800GT nVidia card. The particle filter uses here 1000 particles.

Fig. 3 presents the tracking results in a synthetic and a real case. The estimation is overprinted in wireframe. After a few iterations, it can be noticed that the filter delivers an estimation which efficiently tracks the cube and the polyhedral object. Despite the use of raw data provided by the graphics card, the mean running times for one iteration are much lower than one second (0.15s / 0.27s depending on the size of the ROI) which proves the efficiency of the architecture for this kind of problem. Indeed, the particle methods are known to be very expensive in terms of running time. To our best knowledge, few works exhibit real-time performances, especially by using realistic object models (described by several thousands of polygons).

6. Conclusion

We presented a model-based algorithm to track 3D objects. The strength of this approach lies in a direct estimation of the 3D parameters in the real world using raw images. The computational load of the method is here distributed between the central processing unit and the graphics processing unit of a standard PC. The proposed distributed algorithm allows a real time tracking of complex objects using a sequential Monte Carlo method.

References

[1] P. Lanvin, J.-C. Noyer, M. Benjelloun, "Model Based tracking of 3D object based on a Sequential Monte-Carlo Method", *IEEE 38th Asilomar Conference on Signals, Systems and Computers,* Pacific Grove, CA, USA, November 2004.

[2] P. Lanvin and J-C. Noyer and M. Benjelloun, "Non-Linear Estimation of Image Motion and Tracking", *IEEE International Conference on Multimedia and Expo,* Baltimore, USA, July 2003.

[3] C. Hue, J.-P. Le Cadre, P. Pérez, "A Particle Filter to Track Multiples Objects", *IEEE Workshop on Multi-Object Tracking,* Vancouver, Canada, July 2001.

[4] A. I. Comport, E. Marchand, F. Chaumette, "Robust Model-Based Tracking for Robot Vision", *IEEE Int. Conf. on Intelligent Robots and Systems, IROS'04,* Sedai, Japan, September 2004.

[5] Y. Bar-Shalom and X.-R. Li, *Estimation and Tracking: Principles, Techniques and Software,* Artech House, 1993;

[6] A. Doucet, N. de Freitas, N. Gordon, *Sequential Monte-Carlo Methods in Practice,* Springer Verlag, 2002.

[7] J. Fung, S. Mann, "Computer Vision Signal Processing On Graphics Processing Units", *IEEE ICASSP,* ?, 2004

[8] R. Strzodka, C. Garbe, "Real-time motion estimation and visualization on graphics cards", *In Proceedings IEEE Visualization,* Austin, Texas, October 2004.
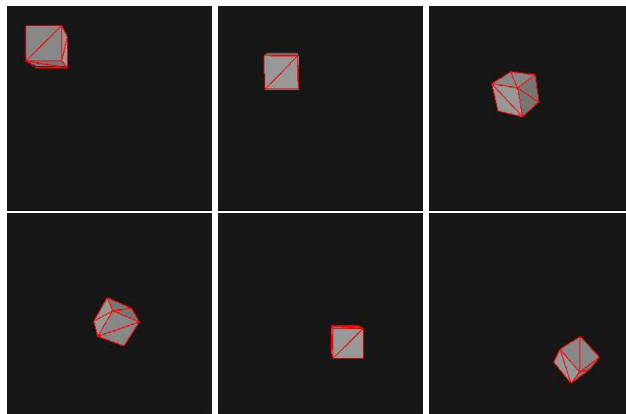
(a) Synthetic sequence (512×512) - 0.15s/frame
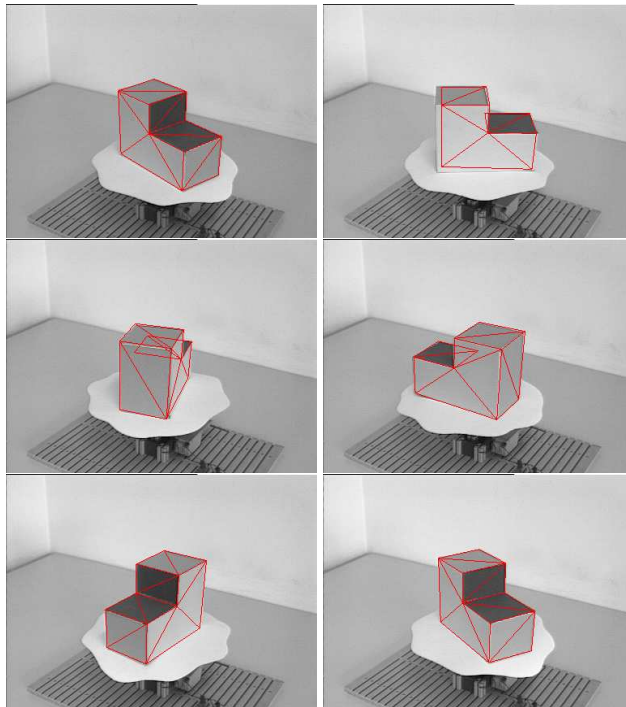
(b) Real sequence (384×288) - 0.27s/frame

Figure 3: Tracking results at several time instants