Dynamic Textures Segmentation with GPU

Juan Manuel Rodríguez, Francisco Gómez Fernández,
María Elena Buemi, and Julio Jacobo-Berlles

Departamento de Computación, Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires, Buenos Aires, Argentina

Abstract. This work addresses the problem of motion segmentation in video sequences using dynamic textures. Motion can be globally modeled as a statistical visual process know as dynamic texture. Specifically, we use the mixtures of dynamic textures model which can simultaneously handle different visual processes. Nowadays, GPU are becoming increasingly popular in computer vision applications because of their cost-benefit ratio. However, GPU programming is not a trivial task and not all algorithms can be easily switched to GPU. In this paper, we made two implementations of a known motion segmentation algorithm based on mixtures of dynamic textures. One using CPU and the other ported to GPU. The performance analyses show the scenarios for which it is worthwhile to do the full GPU implementation of the motion segmentation process.

1 Introduction

Motion and texture are key characteristics for video interpretation. The recognition of textures in motion allows video analysis in the presence of water, fire, smoke, crowds, among others. Understanding these visual processes has been very challenging in computer vision. Some motion segmentation methods are based on optical flow [1,2]. This approach presents difficulties like aperture and noise problems. The classical solution is to regularize the optical flow field, however, this produces unwanted effects in motion, smoothing edges or regions where the movement is smooth (for example, vegetation in outdoor scenes).

To analyze the dynamical visual processes we need a model that can describe them. To fully understand the properties of dynamical visual processing, learning a model, given our measurements (a finite sequence of images), it is necessary to recover the scene that was generated. The recognition of textures in movement based on observed video sequences sampled from stochastic processes taking into account the variations in time and space is called dynamic textures (DT) [3].

Dynamic textures have been used for segmentation of visual processes in video. However, when multiple dynamic textures (possibly superimposed) occur in a same scene, this model is not capable of discriminating them well. To face this problem, a Mixture of Dynamic Textures (MDT) [4] model has been proposed, to handle this issue as a constituent part of the model. The MDT algorithm can classify a set of input video sequences into different categories, given the number

L. Alvarez et al. (Eds.): CIARP 2012, LNCS 7441, pp. 607–614, 2012.

of different visual processes to model. Thus, to segment a single video, we can easily partition it into a group of sub-videos (spatio-temporal patches that fulfill the entire video) and then classify them using MDT. The main drawback using this video segmentation procedure is that the visual process is reduced as a result of the patch-based partition, similar to the aperture problem in optical flow. To address this problem, a global generative model called Layered Dynamic Texture (LDT) [5] has been introduced, which has co-occurring dynamic textures as part of the model. Even considering the new approaches to LDT [5], using them for motion segmentation is slow and will not be part of our study. Implementing these statistical models is computationally demanding. Thus, taking advantage of cutting-edge technology is a necessity. Nowadays, the use of Graphics Processing Units (GPU) in computer vision applications are becoming increasingly popular because of their cost-benefit ratio and their suitability to general purpose computing. In this work, we implement two versions of a motion segmentation algorithm based on the MDT model [5]: one using a traditional CPU processor and other using a GPU-translated optimized implementation with library modules. Computing time tests are carried out to evaluate the benefits of porting to this technology. No quality tests were performed; those were already exposed in previous publications, and the results we obtained are the same here. The structure of this paper is as follows: section 2 describes the dynamic textures and the mixture dynamic models, section 3 shows the implementation developed in this paper, in section 4 we discuss the testing results for dynamic texture segmentation in videos with mixtures of dynamic textures in CPU and GPU. Finally, in section 5 we present the conclusions and future work.

2 Dynamic Texture Model

A dynamic texture [3] is a generative model for both the appearance and the dynamics of video sequences. It consists of a random process containing an observed variable y_t, which encodes the appearance component (video frame y at time t), and a hidden state variable x_t, which encodes the dynamics (evolution of the video over time). The state and observed variables are related through the LDS (Linear Dynamical System) defined by:

$$\begin{cases} x_{t+1} = Ax_t + v_t \\ y_t = Cx_t + w_t \end{cases} \tag{1}$$

where $x_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$ (typically $n \leq m$). The parameter $A \in \mathbb{R}^{n \times n}$ is a state-transition matrix, and $C \in \mathbb{R}^{m \times n}$ is an observation matrix. The driving noise processes v_t and w_t are normally distributed with zero mean and covariances Q and R respectively, that is, $v_t \sim \mathcal{N}(0, Q)$ and $w_t \sim \mathcal{N}(0, R)$. This model is extended to an initial state x_1 of arbitrary mean μ and arbitrary covariance S, that is, $x_1 \sim \mathcal{N}(\mu, S)$. This extension produces a richer video model that can capture variability in the initial frame and is necessary for learning a dynamic texture from multiple video samples with different initial frames (as is the case in

clustering and segmentation problems). The dynamic texture is specified by the parameters $\Theta = \{A, Q, C, R, \mu, S\}$. It can be shown [6] from this definition that the distributions of the initial state, the conditional state, and the conditional observation are:

$$p(x_1) = G(x_1, \mu, S), \quad p(x_t|x_{t-1}) = G(x_t, Ax_{t-1}, Q), \quad p(y_t|x_t) = G(y_t, Cx_t, R)$$

where $G(x, \mu, \Sigma) = (2\pi)^{-n/2}|\Sigma|^{-1/2}e^{-\frac{1}{2}\|x-\mu\|^2_\Sigma}$ is the n-dimensional multivariate Gaussian distribution, and $\|x\|^2_\Sigma = x^T\Sigma^{-1}x$ is the Mahalanobis distance with respect to the covariance matrix Σ. Let $x_1^\tau = (x_1, \cdots, x_\tau)$ and $y_1^\tau = (y_1, \cdots, y_\tau)$ be the sequence of states and the sequence of observations, respectively. Then, their joint distribution is $p(x_1^\tau, y_1^\tau) = p(x_1)\prod_{t=2}^{\tau} p(x_t|x_{t-1})\prod_{t=1}^{\tau} p(y_t|x_t)$.

2.1 Mixtures of Dynamic Textures

Under the mixtures of dynamic textures model, an observed video sequence y_1^τ is sampled from one of K dynamic textures, each having some nonzero probability of occurrence. This is a useful extension for two classes of applications. The first class involves a video that is homogeneous and the second class involves an inhomogeneous video, that is, a video composed of multiple processes that can be individually modelled as a dynamic texture with different parameters. In this model, the random variable $z \sim multinomial(\alpha_1, ..., \alpha_K)$ with $\sum_{j=1}^{K}\alpha_j = 1$ indicates which of the K dynamic textures, each with probability α_j of occurrence, is used to represent the observed sequence. The model parameters are given by $\{\Theta_1, \ldots, \Theta_K\}$, where $\Theta_i = \{A_i, Q_i, C_i, R_i, \mu_i, S_i\}$ as described before. The probability of y_1^τ is given by:

$$p(y_1^\tau) = \sum_{j=1}^{K} \alpha_j p(y_1^\tau|z = j),$$

where $p(y_1^\tau|z = j)$ is the class conditional probability of the j-th dynamic texture, i.e., the texture component parametrized by $\Theta_i = \{A_i, Q_i, C_i, R_i, \mu_i, S_i\}$. Fig. 1 shows the MDT as a graphical model.

2.2 Parameter Estimation

The parameters of K dynamic textures to fit a set $\{y^{(i)}\}_{i=1}^{N}$ of N video sequences, are estimated using the Expectation-Maximization (EM) algorithm [7], that is, an iterative procedure for estimating the parameters of a probability distribution, given that the distribution depends on hidden variables. For the mixtures of dynamic textures model, the observed data is a set of video sequences $\{y^{(i)}\}_{i=1}^{N}$, and the missing information consists of: 1) the assignments $z^{(i)}$ of each sequence to a mixture component, 2) the hidden state sequence $x^{(i)}$ that produces $y^{(i)}$, and 3) the parameter vector $\Theta = \{\Theta_j\}_{j=1}^{K}$. Each iteration of the EM algorithm consists of two steps:

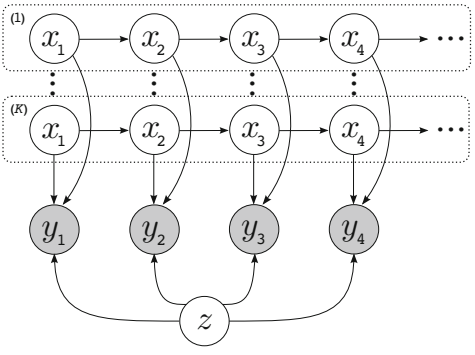

Fig. 1. Graphical model for mixtures of dynamic textures

- **E-step**: $\mathcal{Q}(\Theta;\hat{\Theta}) = \mathbb{E}_{X,Z|Y;\hat{\Theta}}(\log p(X,Y,Z;\Theta))$
- **M-step**: $\hat{\Theta}^* = \arg\max_{\Theta} \mathcal{Q}(\Theta;\hat{\Theta})$

Step E estimates hidden states, and hidden assignment variables with the current parameters, and step M computes new parameters given the previous estimates. Function $p(X,Y,Z;\Theta)$ is the complete-data likelihood of the observations, hidden states, and hidden assignment variables, parametrized by Θ. In mixtures of dynamic textures, the training data are considered to be a set of independent video sequences. In [4] the authors present EM for the mixtures of dynamic textures algorithm. In this method, the E-step relies on the Kalman smoothing filter to compute: 1) the expectations of the hidden state variables x_t, given the observed sequence $y^{(i)}$ and the component assignment $z^{(i)}$, and 2) the likelihood of observation $y^{(i)}$ given the assignment $z^{(i)}$. Then, the M-step computes the maximum-likelihood parameter values for each dynamic texture component j by averaging over all sequences $\{y^{(i)}\}_{i=1}^{N}$, weighted by the posterior probability of assigning $z^{(i)} = j$. The initialization of the EM algorithm is done setting each Θ_j using the method in [3] on a random video sequence from the training set.

3 Video Classification and Motion Segmentation

Mixtures of dynamic textures (MDT) are well suited to motion segmentation, where a moving object or a group of them can be characterized by a dynamic texture. If the model of a dynamic texture is known, then one can estimate the probability of an observed sequence y is generated by the model. In the context of MDT, given a set of video sequences $\{y_i\}$, once the MDT model is learned for $\{y_i\}$ (i.e. all parameters Θ of K dynamic dextures are estimated), each sequence y_i can be classified as the j-th mixture component with the largest posterior probability of being generated by j.

$$\ell_i = \arg\max_j \left(\log p(y^{(i)};\Theta_j) + \log \alpha_j \right) \tag{2}$$

This procedure automatically performs a classification of a video dataset into K categories, useful for video retrieval or video semantics. The aim of motion segmentation is to create a static image describing the regions from a video with homogeneous appearance and motion, i.e. annotate each video location with the number of component to which it belongs, such as fire, water, crowd, traffic jam, etc. This can be achieved generating a set of spatio-temporal patches from a single video, then classifying them using MDT as mentioned before, and finally extracting a new patch at each pixel location and assigning it to a mixture component, according to Equation (2). Algorithm 1 summarizes the motion segmentation process.

Algorithm 1. Motion segmentation with MDT

Input: video y, number of components K
Output: segmentation image M

1. Extract N non-overlapping spatio-temporal patches $\{y^{(i)}\}_{i=1}^{N}$ from input video y
2. Call EM algorithm with $\{y^{(i)}\}_{i=1}^{N}$ and K
3. For each p pixel location in y
   3.1. Let $y^{(i)}$ a spatio-temporal patch centered at p
   3.2. Let ℓ_i given by Equation (2), Set $M_p \leftarrow \ell_i$
4. Return matrix M segmented into K components

4 Results

This section presents computer time performance results of the MDT algorithm presented in the previous section. Both CPU and GPU implementations were developed in order to compare each other. Most important parameters and video frame size were tried to test their impact in the execution time.

To carry out time performance tests, the Synthetic Dynamic Texture Segmentation Database (SynthDB) [8] was used. This database is composed of 299 synthetic videos especially generated to assess video segmentation of visual processes. Each video is composed of K components ($K = \{2, 3, 4\}$), of different temporal textures like water, fire, moving plants, etc. Also, they contain ground-truth template and an initial segmentation contour. Video dimensions are 160×110 with 60 frames at 30 fps.

Algorithm implementations were tested on a CPU computer with an AMD Phenom processor and an NVIDIA Tesla C2070 1.15 GHz GPU. The NVIDIA Tesla C2070 has 448 streaming processor cores and a total amount of global memory of 5376 MBytes. The CPU implementation was developed using Matlab and translated into GPU using functions built in the Parallel Computing Toolbox.

To perform the tests, 3 runs of the entire segmentation process were done over 3 different videos (and the one with minimum time was selected), for 3 different components $K = \{2, 3, 4\}$ and 4 different resized videos (when memory limitations were satisfied). Resized videos were generated scaling the original

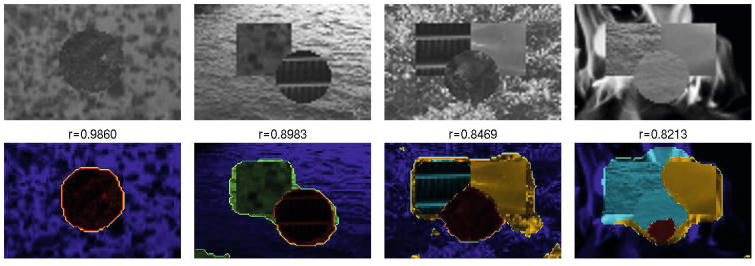

Fig. 2. Video frame examples from SynthDB with $K = \{2, 3, 4, 4\}$, respectively (first row). Their segmentation (best viewed in color) and Rand index r is shown below.

image sequence ($160 \times 110 \times 60$) by a scaling factor of $SF = \{0.5, 1, 1.5, 2\}$, obtaining in this way, for each K, 3 more videos of 80×55, 240×165, and 320×220 all with 60 frames.

Fig. 2 shows segmentation examples over SynthDB. The segmentation quality is computed with the Rand index metric [9]. Following [10], in order to avoid the bottleneck of computing the inverse of a covariance matrix in the Kalman Filter at the expectation step, a C++ implementation [11] of the Cholesky factorization using GPU [12] and CUBLAS [13] was used. This implementation is based on the fact that, as covariance matrices are positive-definite, it is possible to perform the Cholesky factorization and then solve the (upper and lower) triangular systems using TRSM (Triangle Solve Multiple) function of CUBLAS.

Fig. 3 shows computing times with and without the GPU's inverse implementation described below. Darker bars show GPU times. We ran the tests using videos scaled using $SF = \{0.5, 1, 1.5, 2\}$ and $K = \{2, 3, 4\}$. It can be observed a reduction in computing times for large videos. This is due to the fact that the ratio of data transfer time (to and from GPU) with respect to GPU processing time is lower for larger videos. For smaller videos, computing times are better for CPU processing than for GPU processing.

Fig. 4 shows the GPU performance against CPU for all the segmentation process with $K = \{2, 3, 4\}$ and scaling factor $SF = \{0.5, 1, 1.5\}$, and $SF = 2$ only for $K = 2$ due to GPU memory limitations.

Using the same argument as before, the overhead in time of switching data back and forth to GPU has to be lower than the processing time. Therefore, as expected, the GPU implementation outperforms the CPU implementation only when the video is bigger than the original size (1.5 and 2 times bigger).

Finally, most important parameters and video frame size were tried to assess their impact in the execution time. As expected, frame size, sub-video size m and number of components to find K affect directly the computer execution time. However, varying the size n of the hidden state x and fixing the remaining parameters, computer times keep on approximately constant. Also, it was found that the best parameters are $m = 100$, $n = 10$ with respect to Rand index and execution time.

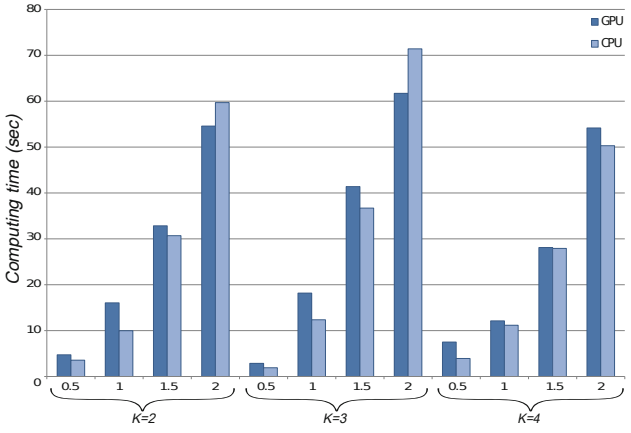

Fig. 3. Time comparisons of two implementations of the inverse using CPU and GPU. Horizontal axis show different scaling factor grouped by segmentation component K.

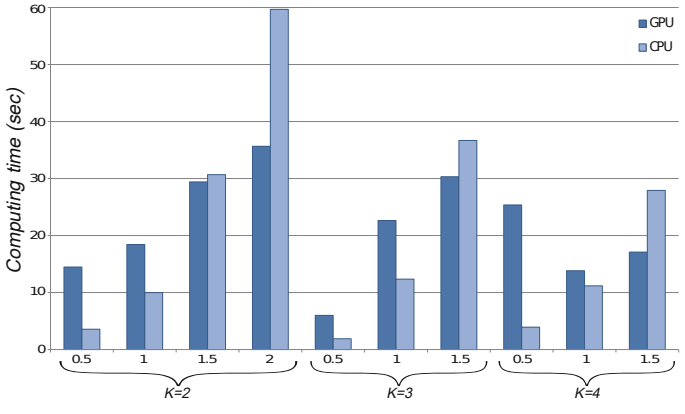

Fig. 4. Computing time performances of the segmentation process on GPU and CPU. Horizontal axis show different scaling factors grouped by segmentation components K.

5 Conclusions and Future Work

In this work we presented CPU and GPU implementations of the motion segmentation algorithm described in [4]. Our GPU implementation is an adaptation from the CPU implementation using library modules that run directly over a GPU card. The importance of this work resides in the analysis of the cases where making a full GPU implementation is a worthwhile work. Performance tests were carried out in order to evaluate benefits and drawbacks of porting algorithms to GPU. Our performance evaluation showed that computing time is reduced significantly with the use of GPU when video size equals or exceeds 320×240, which is the case for video applications of most practical interest.

When analyzing the bottleneck of the MDT algorithm, matrix operations and specially computing the inverse, are the most time-consuming tasks. Our results showed that a meaningful acceleration can be achieved with the use of a specialized GPU function to compute the inverse.

As future work, we are interested in implementing the segmentation algorithm based on layered dynamic textures [14]. This showed better quality results but has a more complex model. Another line of work is to split the learning part of the algorithm from the segmentation part, in order to achieve real time motion segmentation. In this approach, the effort would be put on making the segmentation part online, leaving the learning part as an offline process.

References

1. Horn, B.K.P., Schunck, B.G.: Determining optical flow. Artificial Intelligence 17(1-3), 185–203 (1981)
2. Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: IJCAI 1981, pp. 674–679 (1981)
3. Doretto, G.: Dynamic textures: Modeling, learning, synthesis, animation, segmentation, and recognition, Thesis (Ph.D.)–University of California, Los Angeles
4. Chan, A.B., Vasconcelos, N.: Modeling, clustering, and segmenting video with mixtures of dynamic textures. PAMI 30 (May 2008)
5. Chan, A.B., Vasconcelos, N.: Variational layered dynamic textures. In: CVPR, pp. 1062–1069 (June 2009)
6. Roweis, S., Ghahramani, Z.: A unifying review of linear Gaussian models. Neural Comput. 11(2), 305–345 (1999)
7. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. J. Roy. Statist. Soc. Ser. B, Meth. 39(1), 1–38 (1977)
8. Chan, A.: Synthetic Dynamic Texture Segmentation Database (July 2009), `http://www.svcl.ucsd.edu/projects/motiondytex/db/dytex_synthdb.zip`
9. Hubert, L., Arabie, P.: Comparing partitions (1985)
10. Huang, M.-Y., Wei, S.-C., Huang, B., Chang, Y.-L.: Accelerating the Kalman Filter on a GPU. In: ICPADS (2011)
11. Bouckaert, R.: Matrix inverse with Cuda and CUBLAS, `http://www.cs.waikato.ac.nz/~remco/`
12. Ltaief, H., Tomov, S., Nath, R., Du, P., Dongarra, J.: A Scalable High Performant Cholesky Factorization for Multicore with GPU Accelerators. In: Palma, J.M.L.M., Daydé, M., Marques, O., Lopes, J.C. (eds.) VECPAR 2010. LNCS, vol. 6449, pp. 93–101. Springer, Heidelberg (2011)
13. NVIDIA, Cuda cublas library, Version 4.1 (January 2012)
14. Chan, A.B., Vasconcelos, N.: Layered Dynamic Textures. IEEE Transactions on Pattern Analysis and Machine Intelligence 31, 1862–1879 (2009)