Planar tracking using the GPU for augmented reality and games*

Luis A. Leiva† Antonio Sanz‡ José M. Buenaposada§
Universidad Rey Juan Carlos (MADRID, SPAIN)

1 Introduction

Planar tracking on video streams is a subject of interest in the augmented reality, robotics, human computer interaction for games or computer vision fields. Traditional approaches to tracking are based on finding correspondences in successive images. This can be achieved by computing optical flow or by matching a sparse collection of features. The tracking method presented in this paper belongs to the first group of methods. It is based on minimizing the sum-of-squared differences (SSD) between a previously stored image of the target (the template) and the current image of it.

There are two efficient SSD-based planar tracking algorithms [Baker and Matthews 2004]: Hager and Belhumeur's Jacobian Factorization and Baker and Matthews' Inverse Compositional Image alignment (ICIA). Both algorithms can achieve real-time performance on the CPU with a relatively small size of the template (100×100 pixels). On the other hand, they can not maintain real-time performance with larger templates as the precomputed matrices used on tracking do not fit on the CPU cache memory. The aim of this paper is to show that this kind of algorithms can be accelerated on the GPU even for the case of large templates or using more iterations and resolution levels on tracking. We have implemented the ICIA algorithm on the GPU obtaining impressive performance.

2 Method overview

The same ICIA algorithm can be used with translation, rotation-translation-scale, affine or homography motion models. In our current GPU implementation we describe the target motion using four parameters, $\boldsymbol{\mu} = (t_u, t_v, \theta, s)$, corresponding to rotation, translation and scale applied to the pixel coordinates, $\mathbf{x} = (u, v)^\top$, of the template. The motion model function is given by $f(\mathbf{x}, \boldsymbol{\mu}) = (s+1)\mathtt{R}(\theta)\mathbf{x} + \mathbf{t}$, where $\mathbf{t} = (t_u, t_v)^\top$ and $\mathtt{R}(\theta)$ is a 2D rotation matrix.

The algorithm assumes known motion parameters $\boldsymbol{\mu}_t$ for the image captured at time instant t. On time instant $t+1$ the algorithm performs Gauss-Newton iterations on the captured image to estimate the target motion from $\boldsymbol{\mu}_t$. The steps for each iteration are: 1) warp the current image $I(\mathbf{y}, t+1)$ with motion parameters from previous iteration, $\boldsymbol{\mu}_{t,j}$ (j is the iteration index), computing the rectified image $I(f(\mathbf{x}, \mu_{t,j}), t+1)$; 2) compute the error vector $\mathcal{E}$ subtracting the template from the rectified image; 3) compute the motion parameters increment from last iteration, $\delta\boldsymbol{\mu}_{t,j} = \mathtt{M}^\dagger\mathcal{E} = (\mathtt{M}^\top\mathtt{M})^{-1}\mathtt{M}^\top\mathcal{E}$, where $\mathtt{M}$ is the constant Jacobian matrix that establish how the template grey levels varies w.r.t. motion parameters;

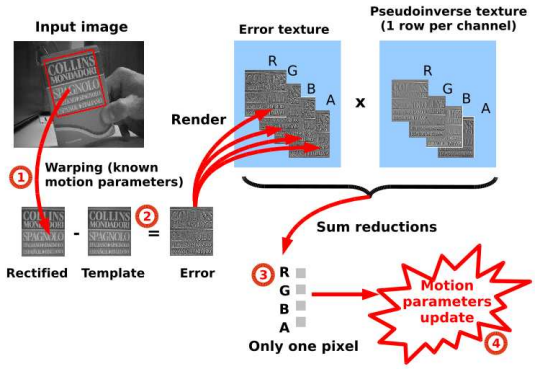

Figure 1: *Tracking algorithm.*

4) compute the new motion parameters for next iteration, $\boldsymbol{\mu}_{t,j+1}$, using the relation $f(\mathbf{x}, \boldsymbol{\mu}_{t,j+1}) = f(f^{-1}(\mathbf{x}, \delta\boldsymbol{\mu}_{t,j}), \boldsymbol{\mu}_{t,j-1})$ (see [Baker and Matthews 2004]).

Steps 1, 2 and 3 of the algorithm have been implemented on the GPU (see Fig.1) whereas the step 4 has been implemented on the CPU (note that $\delta\boldsymbol{\mu}$ is a 4-element vector in this case). In step 1, we apply the captured image and the template (both of the same size) to a quad (corresponding corners are defined by $f(\mathbf{x}, \boldsymbol{\mu}_{t,j})$), step 2 is implemented on a pixel shader that makes the texture difference (between the template and the rectified image from step 1). Step 3 consists of the multiplication of a constant matrix (the Jacobian matrix pseudoinverse) by the error vector. We have implemented it on a shader that performs element by element texture multiplication. One texture is filled with the Jacobian pseudoinverse rows, constant all over the process, and the other one is filled with the computed error vector. Finally, we perform a sum reduction to get the four real values of the motion update vector, $\delta\boldsymbol{\mu}_{t,j}$.

3 Conclusion

A real-time planar region tracker is very useful in applications like augmented reality. On the other hand the SSD algorithms, like the ICIA, are simply not practical on CPU using templates beyond 100x100 pixels. The CPU results on an Intel Centrino 1,86 GHz is 4 fps for the 256x256 pixels template and less than a 1 fps using a 512x512 pixels template. On a GPU implementation we get impressive results (all for 10 iterations per resolution level and 2 resolution levels tracking, like the CPU tests) for a 256x256 pixels template: NVIDIA 8800GTS 96 fps, NVIDIA 7800GTX 62 fps, NVIDIA 6800GT 43 fps; and for 512x512 pixels template: NVIDIA 8800GTS 55 fps, NVIDIA 7800GTX 27 fps and NVIDIA 6800GT 18 fps. Those results are very promising and we plan to implement on GPU other planar tracking algorithms or even a 3D morphable model-based tracking algorithm.

References

BAKER, S., AND MATTHEWS, I. 2004. Lucas-kanade 20 years

*Accepted as a SIGGRAPH 2007 POSTER

†e-mail: la.leiva@alumnos.urjc.es
‡e-mail: antonio.sanz@urjc.es
§e-mail: josemiguel.buenaposada@urjc.es

on: A unifiying framework. *International Journal of Computer Vision (IJCV) 56*, 3, 221–255.