Planar tracking using the GPU for augmented reality and games

Luis Alexis Leiva, Antonio S. Montemayor, José M. Buenaposada
la.leiva@alumnos.urjc.es; {antonio.sanz, josemiguel.buenaposada}@urjc.es

SIGGRAPH2007 FACE TOMORROW

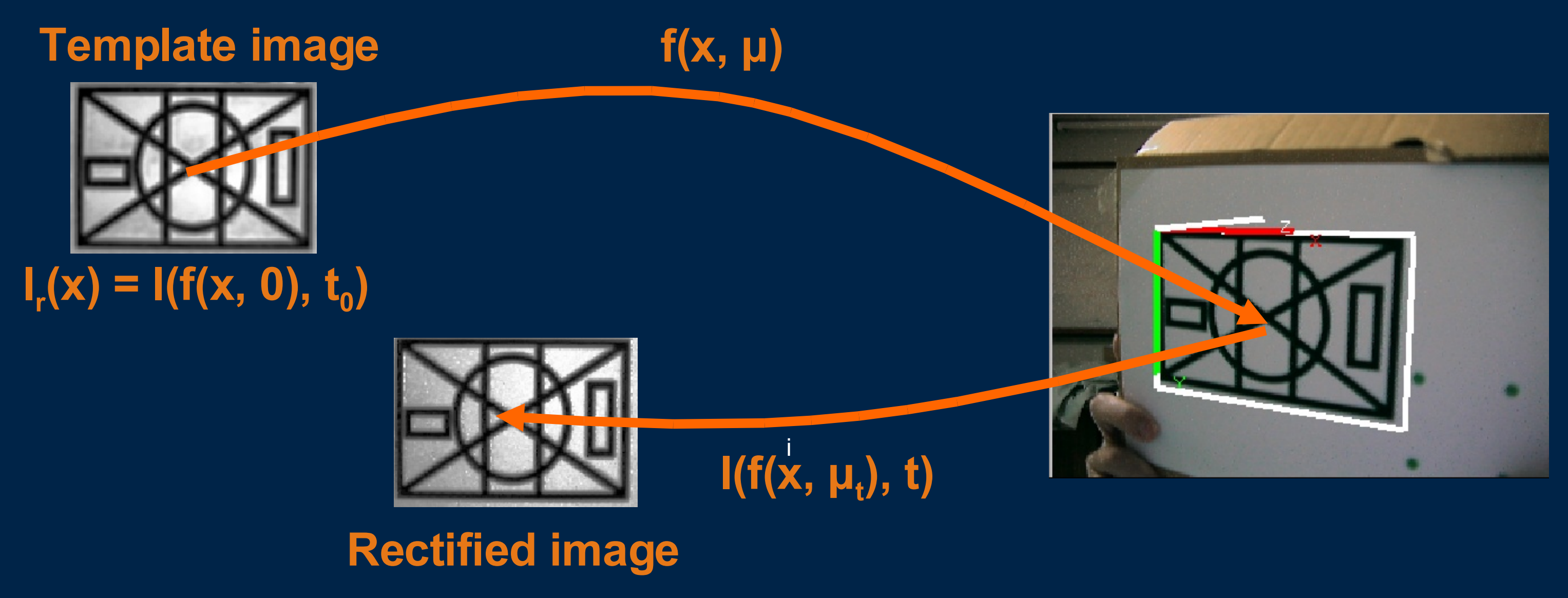
Abstract

In this paper, we present a work-in-progress towards the efficient implementation of planar tracking algorithms on the GPU. The purpose of this work is to locate in 2D, or even 3D, a real planar patch moving in front of the camera by processing a video stream on the GPU. The GPU implementation helps to the tracking algorithm in getting real-time performance with any practical size of the object image template.

Introduction

Planar tracking on video streams is a subject of interest in 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 our work 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:

$$\boxed{\mu_t = \min \| I(f(x, \mu_t), t) - I_r(x) \|^2}$$



There are two efficient SSD-based planar tracking algorithms: 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 (max 100×100 pixels). 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 for the Rotation-Translation-Scale (RTS) motion model:

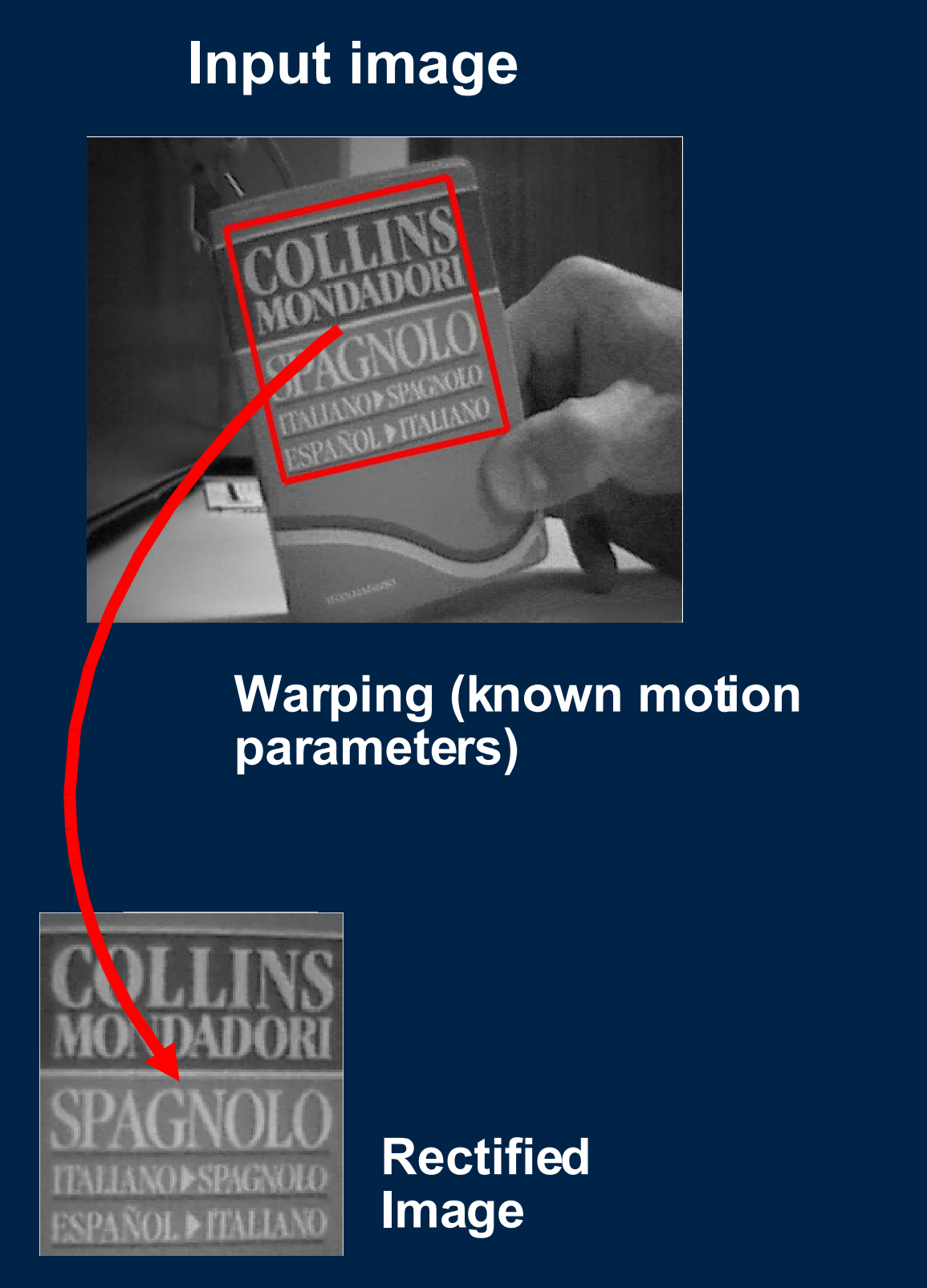
$$f(x,\mu) = (s+1)R(\theta)x + \begin{pmatrix} t_u \\ t_v \end{pmatrix} \quad \text{and} \quad \mu = \begin{pmatrix} s & \theta & t_u & t_v \end{pmatrix}$$

Tracking a planar patch on the GPU

1. Image rectification (warping) stage

In this stage the incoming video image is warped, using the motion parameters from previous iteration to build the rectified image, **I(f(x, μ_t), t+1).**

Taking advantage of the higher GPU memory bandwidth, the image warping is performed by texturing a quad with the incoming video image. The corresponding corners over the captured image (*texcoords*) are defined by f(x, μ_t).



2. Error vector computation

On this stage we substract the template from the rectified image to compute the error vector:

$$\boxed{\varepsilon = I(f(x, \mu_t), t+1) - I_r(x)}$$

We have implemented this step on a fragment shader that makes the texture difference (between the template and the rectified image from step 1).

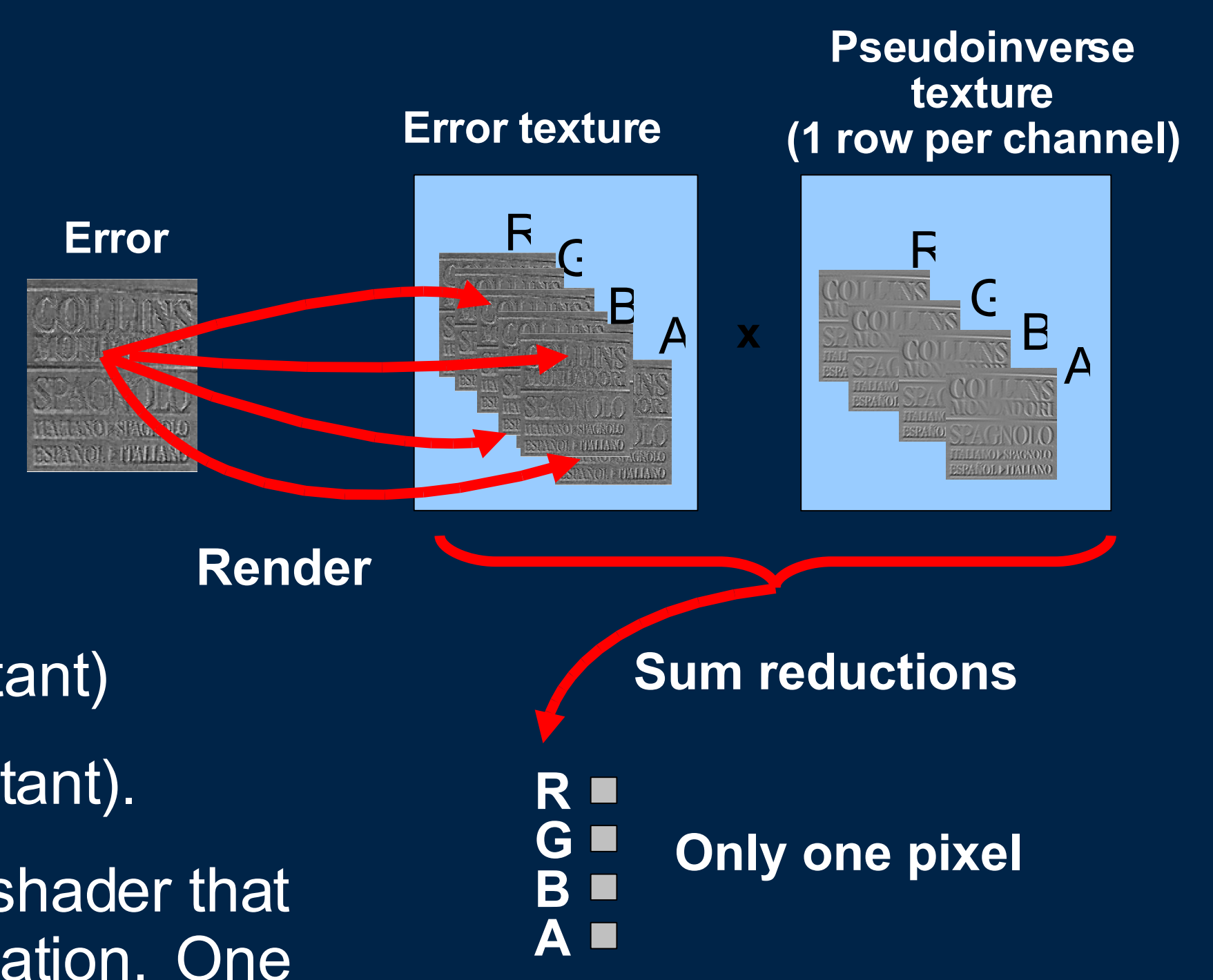
3. Motion parameters increment computation

The minimisation of the SSD cost function, on ICIA algorithm, is done by Gauss-Newton iterations. On each iteration we compute the motion parameters increment, δμ, using:

$$\boxed{\delta\mu = (M^T M)^{-1} M^T \varepsilon = M^* \varepsilon}$$

- M is the Jacobian of grey levels w.r.t. μ (constant)
- M* is the pseudoinverse of the Jacobian (constant).

We have implemented this step on a fragment 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.



4. ICIA motion parameters update

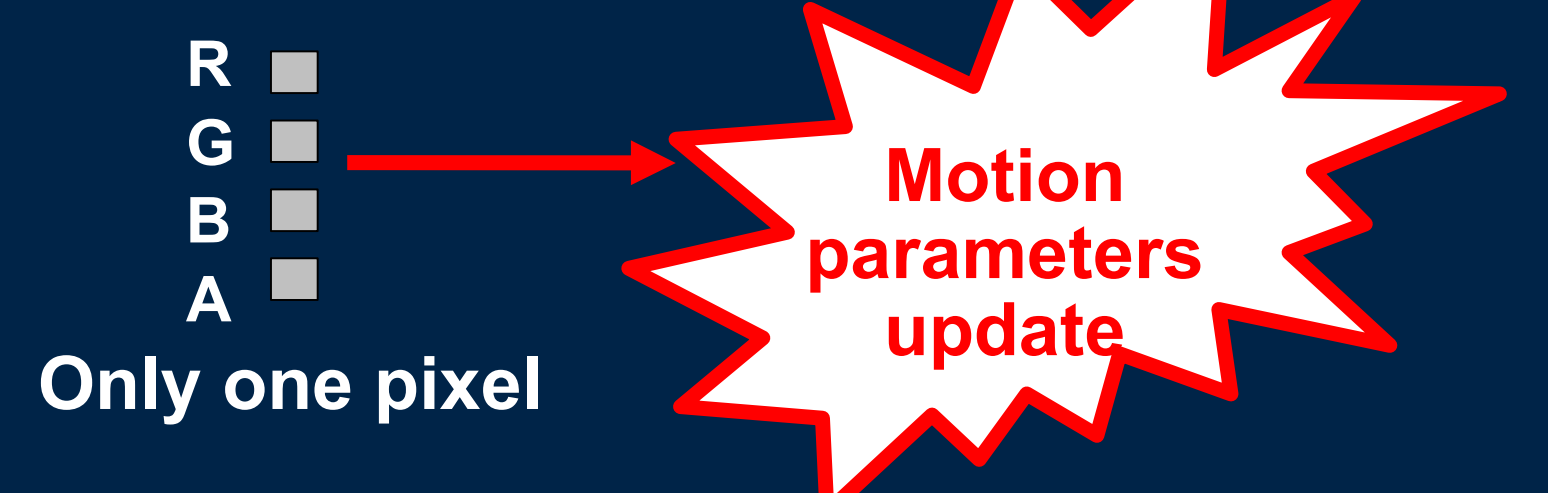
On the last step of the algorithm we compute the new motion parameters for the next iteration, μ_{t+1}, using the compositional relation:

$$\boxed{f(x, \mu_{t+1}) = f(f^{-1}(x, \delta\mu), \mu_t)}$$

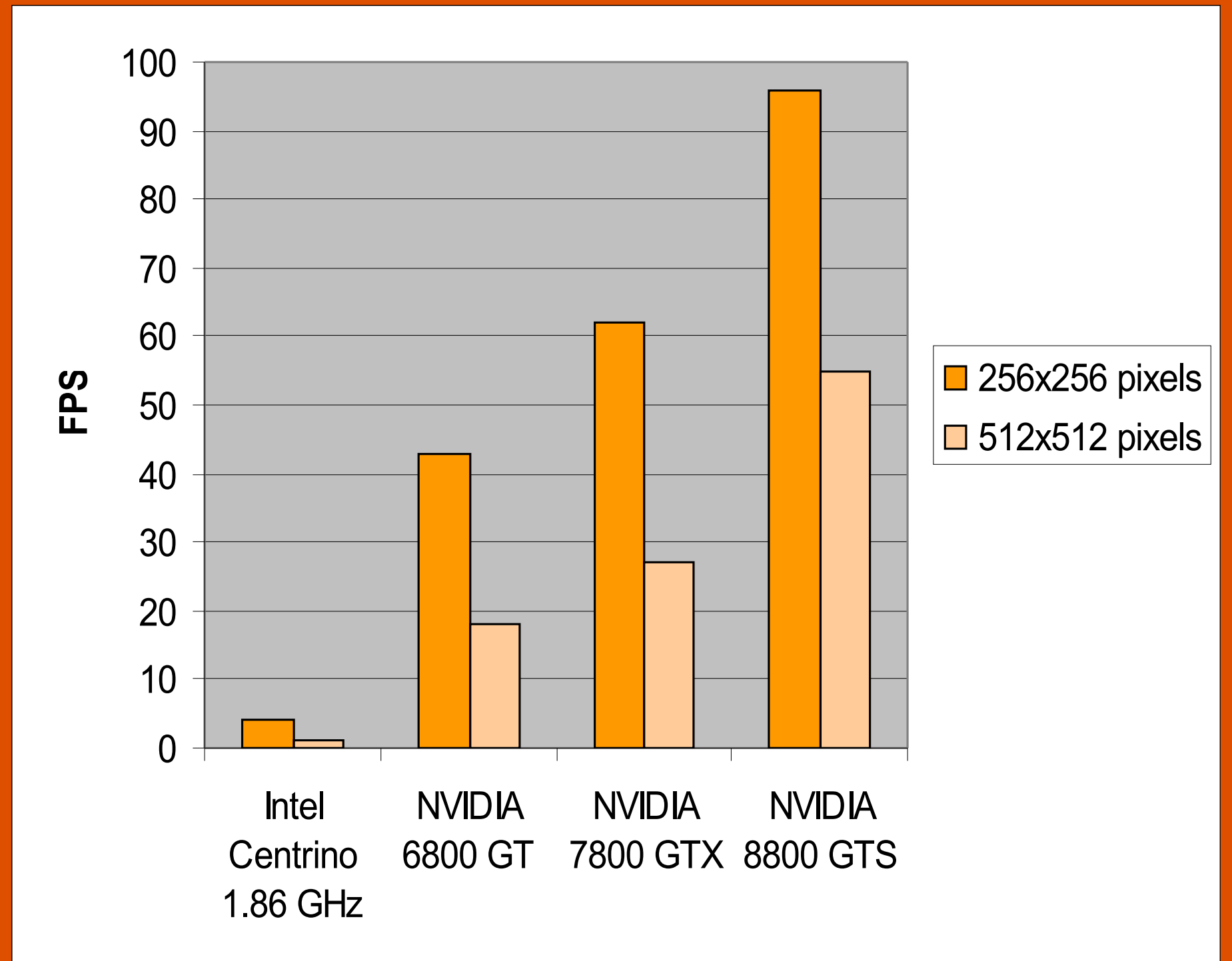
In our case we use the RTS motion model and μ_{t+1} is computed on the CPU as:

$$s_{t+1} = \frac{s_t + 1}{\delta s + 1} - 1 \qquad \begin{pmatrix} t_u \\ t_v \end{pmatrix}_{t+1} = \begin{pmatrix} t_u \\ t_v \end{pmatrix}_t - (s_{t+1}+1)R(\theta_{t+1})\begin{pmatrix} \delta t_u \\ \delta t_v \end{pmatrix}$$

$$R(\theta_{t+1}) = R(\theta_t)R^T(\delta\theta)$$



Experimental Results



Inverse Compositional Algorithm 256x256 and 512x512 pixels templates, 10 iterations per resolution, 2 resolution levels (Rotation-Translation-Scale motion model).

Input video sequences are in QCIF format (320x240 pixels)

Conclusions

A real-time planar region tracker is very useful in applications like augmented reality or interactive games using a video camera.

On the other hand the SSD tracking algorithms, like the ICIA, are simply not practical on CPU using templates beyond 100x100 pixels.

The results we got are very promising and we are planning to implement on GPU other planar tracking algorithms or even a 3D morphable model-based tracking algorithm.