Proceedings of the 10th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2010
27–30 June 2010.

A Low Cost Virtual 3D Human Interface Device using an Optical Flow Algorithm and GPUs

Rafael del Riego[1], José Otero[1] and José Ranilla[1]

[1] *Department of Computer Science, University of Oviedo (Spain)*

emails: rafarfn@gmail.com, jotero@uniovi.es, ranilla@uniovi.es

Abstract

Modern personal computers, including laptops, notebooks and perhaps smartphones, often have a low-resolution camera and a powerful graphic card. In this paper we present a system that uses these resources (camera and GPU) to build a low cost virtual 3D Human Interface Device. To do this, we apply an optical flow algorithm which is characterized by its high degree of parallelization. The experimental results confirm the performance of our system.

Key words: Virtual 3D-HID, GPU, Optical Flow Algorithms.

1 Introduction

Today, personal computers, laptops, notebooks and smartphones have an integrated low-resolution camera (high-resolution in the case of smartphones). Besides, most of them have a graphic processing unit (GPU), a specialized processor that offloads 3D/2D graphics rendering from the microprocessor.

A new computing parading is to use a GPU as a stream processor. This concept turns the massive floating-point computational power of a modern graphics accelerators into general-purpose computing power. In certain applications, this allow us to increase the performance in several orders of magnitude compared to a conventional CPU.

Recently, NVIDIA[1] began releasing cards supporting an API extension to the C programming language CUDA (Compute Unified Device Architecture), which allows specified functions from a normal C program to run on the GPU's stream processors. This makes C programs capable of taking advantage of a GPU's ability to operate on large matrices in parallel while still making use of the CPU when appropriate.

Being aware of these capabilities (CUDA compatible GPUs and a low-resolution camera), in this work we present a system that uses these resources to build a Low Cost Virtual 3D Human Interface Device (3D-HID). Users can interact with the environment

[1] http://www.nvidia.com

 ISBN 13: 978-84-613-5510-5

(real 3D world) by simply moving the camera (in the case of lightweight devices such as smartphones) or moving objects (e.g. hand) in the vicinity of the camera (e.g. laptops). In order to do this, we use an optical flow algorithm, which is characterized by its high degree of parallelization.

In order to point out the aim of this paper, we briefly review some aspects that will be considered. Thus, in section 2 we explain the optical flow algorithms. Section 3 is devoted to the built system. The experimental results are showed in section 4 and finally, section 5 summarizes our conclusions.

2 Optical Flow

Optical flow is the 2D vector field projection of the 3D velocities of object points. In Figure 1 a pair of frames of a classic test sequence is shown, with the true optical flow overimposed. As can be seen, the motion of the objects in the scene is well represented by the optical flow.

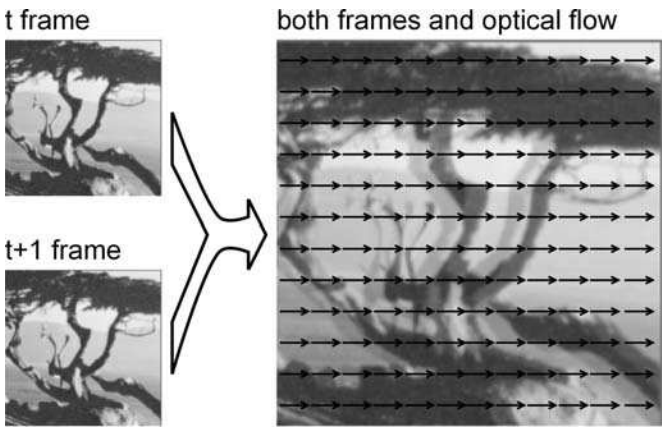

Figure 1: Optical Flow example, from two frames (left) the motion of the scene is measured for each pixel. The resulting vector field (red arrows) is shown in the right side of the figure. Both frames are overimposed.

In the literature, optical flow algorithms are classified in: correlation based techniques, frequency based techniques and gradient based techniques.

Correlation based techniques or block matching algorithms [1] try to maximize a measure of similarity between patches (taken from two consecutive frames) centered in a given pixel. The displacement that maximizes the selected measure divided by the time interval within the acquisition of the frames is the velocity of the pixel (see Figure 2).

Frequency based techniques use a set of tuned spatiotemporal filters to search for the velocity of a pixel [3].

Gradient based techniques use the well known Optical Flow Constraint (OFC) shown in equation 1 in order to compute the optical flow [4].

 ISBN 13: 978-84-613-5510-5

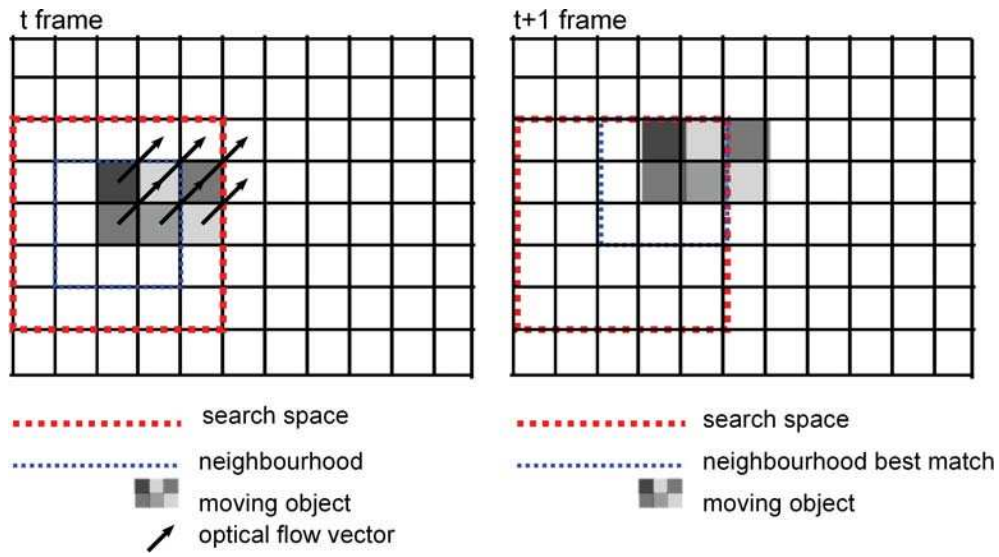

Figure 2: Optical Flow computation using a block matching algorithm. The neighborhood in the first frame (blue dotted square) is found in the second frame in different position. This displacement defines the Optical Flow vector for each pixel.

$$-\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} = \frac{\partial f}{\partial x}u + \frac{\partial f}{\partial y}v = \nabla(f)\cdot\overrightarrow{c} \tag{1}$$

Equation 1 makes the assumption that intensity changes in a sequence of images are only due to the movement of the objects in the scene: a single pixel will have constant brightness in the different positions that it takes during the sequence. Unfortunately, the Aperture Problem (see [5]) states that there is no way to recover the complete optical flow vector using only local (one pixel) information.

In Figure 3 a synthetic example is shown. As can be seen, OFC holds for the selected pixel $(4, 1)$, $(-1, -1)$ velocity verifies the obtained equation: replacing $\frac{\partial f}{\partial x}$ by 1, $\frac{\partial f}{\partial y}$ by 2 and $\frac{\partial f}{\partial t}$ by 3, $u + 2v = -3$ is obtained. Unfortunately, a suitable value that verifies the OFC can be found for one of the components substituting the other by an arbitrary value.

Some authors try to solve the aperture problem with the incorporation of some kind of global information, involving a process of regularization [4]. Some researchers perform a clustering of the OFCs themselves in order to find the most reliable one. Once obtained, the corresponding normal flow to that OFC is obtained [6]. Another alternative is to analyze the measurements in the space of the velocities that is, performing an estimation of the velocity with the results of many systems of OFC equations. Each system of equations is obtained from one pair of pixels in order to estimate the velocity. In this way, the analysis is performed directly in the domain of the data that we want to recover, that is, the u, v space [7, 8, 9].

All the previously approaches are computationally expensive. For example, for a

 ISBN 13: 978-84-613-5510-5

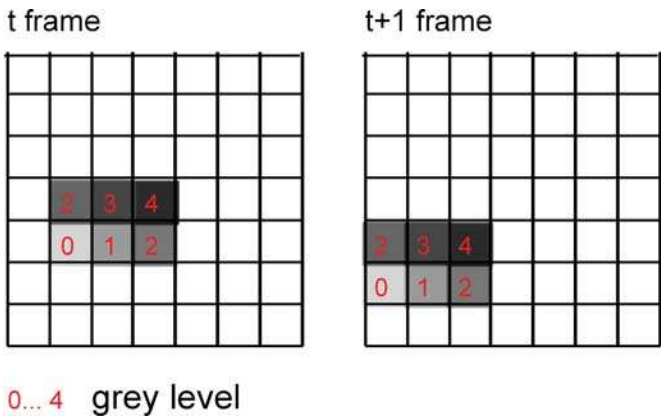

Figure 3: Optical Flow computation using OFC. As can be seen OFC holds for pixel (4,1): $\frac{\partial f}{\partial x} = 1$, $\frac{\partial f}{\partial y} = 2$ and $\frac{\partial f}{\partial t} = 3$, then OFC is $u + 2v = -3$, that holds for $(-1, -1)$.

$n \times m$ pixels image, a search space of $p \times q$ pixels and a neighborhood of $s \times t$ pixels, the number of floating point operations in the case of BMA is $n \times m \times p \times q \times s \times t$. Similar numbers are obtained for gradient based approaches. Because of this, an implementation using a GPU and CUDA speeds up the computation process.

3 The System

System architecture is divided in four main blocks: Video Input, Optical Flow Algorithm, Motion Estimator and Control (see figure 4).

Each subsystem comprises several basic computing units (called *basic-units*). Thus, the execution of several *basic-units* of different images is concurrent (like works pipelined CPUs). This solution is adequate even for single core CPUs; simply the degree of concurrence is smaller.

The communications between *basic-units* use circular buffers. This leads to an increasing of memory consumption but allows more efficient asynchronous execution of the *basic-units*.

Because of the low image resolution, only a smoothing with a small Gaussian kernel is needed in order to decrease the acquisition noise. This benefits the parallel implementation of the whole system because the separability of the filtering.

We will use optical flow field estimation in order to measure/detect the motion in the image sequence acquired with the cameras. The goal is to use the translations in X, Y and Z along with rotation in Z as input signals to the proposed virtual interface.

In order to discriminate the predominant motion in the scene, we use the following operators:

- X and Y translations are measured as the average optical flow in the image: $(X, Y)_T = \frac{\sum_{i=0}^{i=n} \sum_{j=0}^{j=m} (F_x, F_y)_{i,j}}{nm}$, where $(X, Y)_T$ is the traslation vector, $(F_x, F_y)_{i,j}$

 ISBN 13: 978-84-613-5510-5

is the Optical Flow Vector for pixel (i, j), n, m are the number of rows and columns in the image.

- Z translation is measured with the divergence of the optical flow averaged across the image: $Z_T = \frac{\sum_{i=0}^{i=n} \sum_{j=0}^{j=m} \bigtriangledown \cdot (F_x, F_y)_{i,j}}{nm}$ The previous expression is evaluated and averaged for each optical flow value across the whole image.
- Z rotation is measured with the rotational of the optical flow averaged across the image: $Z_R = \frac{\sum_{i=0}^{i=n} \sum_{j=0}^{j=m} \bigtriangledown \times (F_x, F_y)_{i,j}}{nm}$

We implemented two algorithms using CUDA, the proposal in [7] and an hierarchical implementation of Lucas-Kanade algorithm available with OpenCV library [10]. Finally, a bottleneck in Otero et. al. algorithm [7] leads us to choose Lukas-Kanade algorithm [10]. The output of this algorithm is evaluated with the previous operators. The highest output defines the predominant motion in the scene.

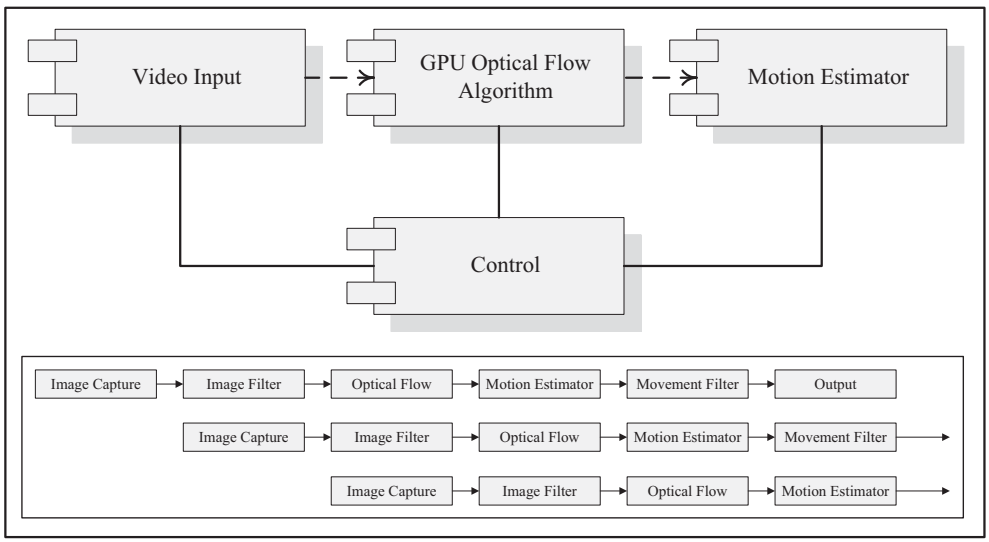

Figure 4: System Architecture

4 Experiments

The hardware setup comprises a laptop and two cameras with different resolution, the usual integrated in the screen frame and an off the shelf usb camera, with the following technical specs:

- CPU AMD Athlon 64 3000+ (1.8 GHz) AM2.
- GPU nVidia GeForce 9500 GT.
- RAM 1024 MB Dual Channel 800 MHz.

 ISBN 13: 978-84-613-5510-5

- Web cam 1 Logitech Quickcam E2500.
- Web cam 2 Logitech Webcam C200.

The system detects easily the motion in different axis, only when velocity module falls below a threshold some errors may appear, mainly due to image acquisition and illumination issues (flickering or low illumination).

Low illumination leads to noise in the image and to the false detection of small movements due to the noisy pixels that appear and disappear. We decided to filter the movements that are below a threshold in order to minimize that error.

During the experiments we found that X and Y translations are easily detected but Z translation is easily detected as X or Y motion, because small displacements in X or Y directions (by the user) lead to relatively high values compared with the divergence of the optical flow field.

When the algorithm was running in the CPU we obtained 17 frames per second, with the performance of other tasks being degraded.

Using CUDA implementations the frame rate increases to 30 frames per second, the hardware limit of the cameras. Standard GPU tasks are not degraded and the CPU can be fully dedicated to other tasks. Thus, the user experience is real-time alike.

Another kind of movement useful as input signal in the virtual interface is rotation in Z axis. The amount of motion cannot be accurately measured but if we use a threshold it can be used to simulate a click.

Summarizing, X and Y translations are correctly detected and measured. Z translations cannot be accurately measured and it is not useful as input signal. Z rotation cannot be accurately measured but a suitable threshold can be used and then, it serves as binary signal.

5 Conclusions

In this work we have showed how to build a Virtual 3D Human Interface Device by using standard resources of the current computers: the integrated camera (or Web cam) and the graphic processing unit (GPU). The system applies optical flow techniques and uses CUDA (Compute Unified Device Architecture, nVidia) to exploit the capabilities of the GPU as stream processor.

X and Y translations are correctly detected and measured by the system. Z translations are detected but not accurately measured and Z rotation cannot be accurately measured but a suitable threshold can be used and then serves as binary signal.

Summarizing, the presented solution is simple, the frame rate is now limited by the resolution of the camera (30 frames per second *vs* 17 in the case of CPU's based solutions), users experience is real-time alike, computer's performance is not being degraded and powerful/additional hardware it is not necessary. Therefore, we have built an efficient and low cost 3D interface.

 ISBN 13: 978-84-613-5510-5

Acknowledgements

We would like to acknowledge the help received from the Spanish Office of Science and FEDER, for their support of the projects TIN2007-61273 and TIN2008-06681-C06-04.

References

[1] P. Anandan, *A computational framework and an algorithm for the measurement of visual motion*, International Journal of Computer Vision **2** (1989) 283–310.

[2] J. L. Barron, D. J. Fleet and S. S. Beauchemin, Performance of optical flow techniques, International Journal of Computer Vision **12(1)** (1994) 43-77.

[3] D. J. Heeger, *Optical flow using spatiotemporal filters*, International Journal of Computer Vision (1988) 279-302.

[4] Berthold K. P. Horn and Brian G. Schunk, *Determining Optical Flow*, In Computer Vision Principles, MIT, Artificial Intelligence Laboratory, 481-497, 1980.

[5] David W. Murray and Bernard F. Buxton, *Experiments in the Machine Interpretation of Visual Motion*, MIT Press, 1990.

[6] P. Nesi, A. del Bimbo and D. Ben Tzvi, *Robust Algorithm for Optical Flow Estimation*, Journal on Computer Vision, Graphics and Image Processing: Image Understanding **61(2)** (1995) 59-68.

[7] J. Otero, A. Otero and L. Sánchez, *Mode based hierarchical optical flow estimation*, MG&V **10(4)** (2001) 489-501.

[8] José Otero, Adolfo Otero and Luciano Sánchez, *3D motion estimation of bubbles of gas in fluid glass, using an optical flow gradient technique extended to a third dimension*, Mach. Vis. Appl. **14(3)** (2003) 185-191.

[9] Brian G. Schunck, *Image Flow Segmentation and Estimation by Constraint Line and Clustering*, IEEE Transactions on Pattern Analysis and Machine Intelligence (1989) 1010-1027.

[10] Jean-Yves Bouguet, *Pyramidal implementation of the lucas kanade feature tracker*, OpenCV Documentation, 2000.

 ISBN 13: 978-84-613-5510-5