Universidade Federal de Pernambuco
Centro de Informática
Bachelor in Computer Engineering

# A Real Time Online Background Subtraction Implementation in CUDA using Incremental Principal Component Pursuit

Djeefther Souza Albuquerque

B. Eng. Dissertation Proposal

Advisor: Veronica Teichrieb
Co-advisor: João Marcelo Teixeira

Recife
September, 2016

**Abstract**

Background Subtraction aims to separate what in a video is foreground (moving objects) from background. It is a very important tool for a lot of applications, such as navigation, surveillance and automatic tools in graphics software. This work intends to extend a previous work made by Rodriguez [1] of an IncPCP (Incremental Principal Component Pursuit) done in MATLAB, by translating it to C++ and then using CUDA to speed up the processing without losing precision. This speedup will allow the real time processing of a video stream, which is a key feature for the applications aforementioned. In the end of this project, the results of our implementation and the reference one will be evaluated and compared regarding accuracy (F-measurement) and speed (FPS (Frames per Second)) using reference ground-truth datasets.

**Keywords:** Background Subtraction, Image Processing, Principal Component Pursuit, CUDA, GPU, Real Time

## Summary

## Background

Background Subtraction aims to separate what in a video is foreground (moving objects) from the background. Figure 1 shows a general view of how it works. A proper background subtraction algorithm should only highlight relevant movement. In the case shown in Figure 1, the water movement is not relevant. For accomplishing that the background model should avoid all undesired changes in the foreground, including movements, light changes, jitter or other kinds of noise.

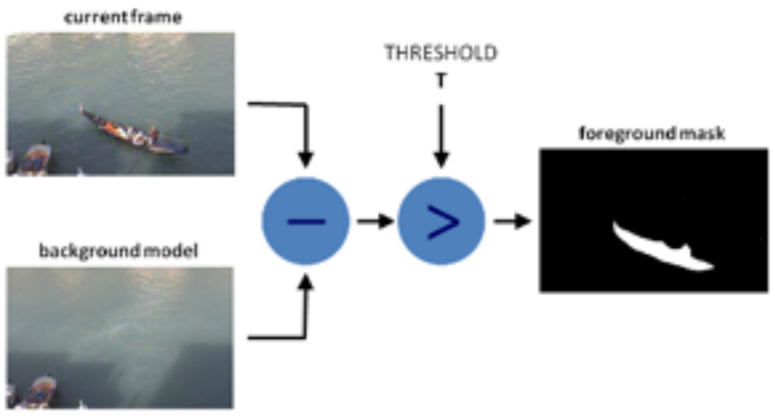

**Figure 1: General scheme for a background subtraction algorithm. Image obtained from (Maghsoumi 2015)**

It is a valuable tool for a lot of applications, such as navigation, surveillance, and automatic tools in graphics software. These applications require a real-time solution. As an example, Figure 2 shows an automatic traffic analyzer that internally uses a background subtraction algorithm to detect cars' movement.

**Figure 2: Example of a background subtraction algorithm in an automatic traffic analyser. Image obtained from (Ferreira 2016)**

This work intends to extend a previous work made by Rodriguez and Wohlberg [1] called IncPCP (Incremental Principal Component Pursuit) by translating the algorithm already implemented in MATLAB® to C++, and then implementing it using CUDA to take benefit of the GPU as coprocessor. This way, we intend to make the algorithm faster and possibly real time, which is relevant to the aforementioned applications, without losing accuracy compared to the original algorithm. In order to do that, the full reading and understanding of their technique and their implementation is an essential part.

PCP (Principal Component Pursuit) is considered the state-of-the-art for video background modelling [1]. It is an algorithm that models the background of a video as a low-rank matrix  where (number of rows, columns and depth or channels, respectively) and n is the number of frames analyzed, as shown in Equation (1). The PCP finds the background and foreground solving the optimization problem in (1). D and S are the video input and foreground, respectively, and they have the same dimensions as L, so and .

(1)

PCP algorithms are used as online and offline methods. The offline method processes an entire video in batch, analyzing all the frames at once. The online method processes each new frame based on a background model and so it could accept video streams. The online method could have a static or adaptive background model. Our implementation corresponds to an online algorithm that fits best the objectives this report has previously mentioned.

The PCP algorithm requires a low-rank evaluation of an SVD (Singular Value Decomposition) [2] as shown in Equations (2) and (3). This significant mathematical operation makes this algorithm timing and memory consuming, since it has to keep in memory a matrix with  floating point numbers as input and compute all of them in an SVD and make an output of  floating point numbers. In an online version of PCP process a full SVD of the last n frames (as n being a memory parameter) is computed for each new frame, without any data reutilization, and just after that SVD computation the rank is reduced by forcing the less significant eigenvalues to zero. This dataflow does not reuse any previous data already computed between frames, and use huge more memory when compared with IncPCP.

(2)

Where for full-size SVD:

- M is the input (complex or real) m × n matrix,
- is an , unitary matrix,
- is an  diagonal matrix with non-negative real numbers on the diagonal, and
- is an  unitary matrix, the conjugate transpose of

Moreover, a fixed rank factorization, which is a partial SVD with rank equal to *r;* is going to be:

(3)

Where for r rank SVD:

- M is the input (complex or real) m × n matrix,
- r is an integer input corresponding to the rank that should be less than or equal to min(m,n),
- is an , unitary matrix,
- is an  diagonal matrix with non-negative real numbers on the diagonal, and
- is an  unitary matrix, the conjugate transpose of

The multiplication of the three output matrices from the full SVD (2) always give the input itself, when the multiplication of three output matrices from (3) always returns an r-rank matrix that depending on the r chosen could be a good approximation or a low rank of the input.

Rodriguez and Wohlberg [1] work made possible to compute an SVD based on a previous SVD from the last frame keeping or increasing the rank in this operation, without the need of recomputing all SVD and then reducing the rank. So it allows us to save memory (working with a low rank matrix all time) as well to save time (reusing already computed data). The input memory usage is  floating point numbers from the current frame, and the output memory usage is going to be floating point numbers considering a 1-rank evaluation is) floating point numbers, which is a a considerable memory saving. They called this mathematical operator as IncSVD (Incremental Singular Value Decomposition). The IncSVD is what makes possible the IncPCP to exist and be faster than previous algorithms.

In the end of this project we aim to compare our C++ version without CUDA, the version in C++ with CUDA optimization and the reference version in MATLAB regarding accuracy (F-measurement [3]) and speed (FPS (Frames per Second)) using reference ground-truth datasets.

The relevance of this technique when compared with the literature is shown for some researches as Yang [4] compares a lot of techniques of background subtraction and introduces its own technique. IncPCP is the faster of the eight techniques and the third better in F-measurement precision.

## Objectives

The main objective of this project is to produce a real time CUDA implementation of IncPCP. In order to do so, a C++ version without CUDA will also be implemented as debug reference. Both of them need the IncSVD operator, as described before, so this operator must also be implemented in both CUDA and C++ platforms. Summing up, this project is going to produce 4 implementation deliverables:

- IncPCP
- CUDA IncPCP
- IncSVD
- CUDA IncSVD

Besides that, all modules need to be tested, debugged and have precision and speed evaluations, in F-measurement tests and FPS, respectively. Tests of the MATLAB reference code will also be performed.

## Schedule

| Activities | September | | | | October | | | | November | | | | December | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Read and Test IncPCP MATLAB code from Rodriguez | | | x | x | | | | | | | | | | | | |
| Read Literature about Background Subtraction and find reference datasets | | | x | x | | | | | | | | | | | | |
| Implement IncSVD in C++ and test its compatibility with MATLAB reference implementation | | | | | x | x | x | | | | | | | | | |
| Implement IncSVD in C++ with CUDA and test its compatibility with MATLAB reference implementation | | | | | | x | x | x | | | | | | | | |
| Implement IncPCP in C++ and test its compatibility with MATLAB reference implementation | | | | | | | | x | x | x | | | | | | |
| Implement IncPCP in C++ with CUDA and test its compatibility with MATLAB reference implementation | | | | | | | | | | x | x | x | | | | |
| Test the general precision, find a good threshold value and general specs of the final implementation | | | | | | | | | | | | | x | x | | |
| Write Dissertation | | | | | | | | | | | | | | | x | x |

## Possible Examiners

There are possible examiners of this project as specified in this propose:

**Abel Guilhermino da Silva Filho**

**Edna Natividade da Silva Barros**

**Hansenclever de França Bassani**

## Signatures

Djeefther Souza Albuquerque
Student

Veronica Teichrieb
Advisor

João Marcelo Teixeira

Co-advisor

## References

1. Rodriguez, P., and Wohlberg, B. 2015a. Incremental Principal Component Pursuit for Video. J Math Imaging Vis (2016) 55:1–18 DOI 10.1007/s10851-015-0610-z https://sites.google.com/a/istec.net/prodrig/Home/en/pubs/incpcp accessed at 21/10/15.
2. “Singular Value Decomposition”, Wikipedia, https://en.wikipedia.org/wiki/Singular_value_decomposition.html [Accessed at 01/04/2016].
3. “F1 score”, Wikipedia, https://en.wikipedia.org/wiki/F1_score accessed at 21/10/15.
4. Y. Hu at el., “An Online Background Subtraction Algorithm using a Contiguously
5. Weighted Linear Regression”, EUSIPCO, Nice, PACA, France, CFP1540S-USB, 2015