Adapting the Sobel Edge Detector and Canny Edge Extractor for iPhone 3GS architecture

Marcelo G. Roque
Computer Science Department
Universidade Federal Fluminense
R. Passo da Pátria, 156, Bloco E, Medialab
Email: dsmarcelo@live.com

Rafael M. Musmanno
Computer Science Department
Universidade Federal Fluminense
R. Passo da Pátria, 156, Bloco E, Medialab
Email: rafa.musmanno@gmail.com

Anselmo Montenegro
Computer Science Department
Universidade Federal Fluminense
R. Passo da Pátria, 156, Bloco E
Email: anselmo@ic.uff.br

Esteban W. G. Clua
Computer Science Department
Universidade Federal Fluminense
R. Passo da Pátria, 156, Bloco E, room 304
Phone: (55 21) 26295646 Email: esteban@ic.uff.br

Abstract—This paper describes the adaptation and optimization of two edge detector algorithms used as low-level operations in the development of augmented reality applications for the iPhone 3GS platform. We have investigated the Sobel and Canny edge detectors and proposed robust solutions for the edge detection problem that can be efficiently used in low processing power devices. We also present here an analysis of the results obtained in the considered platform and which of the optimized edge detectors produced the best overall performance.

Keywords: Sobel Edge Detector, Canny Edge Extractor, iPhone Edge Detectors, Augmented Reality on iPhones.

I. INTRODUCTION

This work presents the adaptation and optimization of classical edge detectors for mobile augmented reality applications, where few computational resources are available.

Augmented reality technology has been used and explored in several contexts and areas, such as education, research, business, and advertising, among others. Nevertheless, a large set of potential applications have been hindered by the fact that the required algorithms are too computationally expensive to be executed on mobile phones and other types of low processing power hardware.

This work presents an approach for edge detection that has been successfully applied to the construction of an augmented reality application in the iPhone 3GS platform.

The full application consists in adding synthetic graphical objects to real world scenes that are captured in real-time by an iPhone 3GS camera. The augmented reality application that was developed is loosely based on the work of Michel Alain [5] [6] and was implemented using *xCode.*

The process begins with the image capture. After this, the captured image passes through a set of image processing stages in order to locate a calibration pattern, added to the scene, which is used to estimate the camera's intrinsic and extrinsic parameters, by using a method based on Tsai's method [11].

Once calibration is carried out, the synthetic objects are inserted right above the calibration pattern in the real scene in the position and orientation of the device's camera.

One of the main points of this work is that camera calibration is a computationally intensive operation itself. Besides, it requires the detection of the image coordinates of a set of feature points in the calibration pattern, whose world space coordinates are previously measured.

The detection of such feature points rely heavily on efficient and robust edge detectors. This has motivated us to search for efficient solutions for edge detection that do not introduce any significant overhead in the overall process of pattern recognition, camera calibration and scene rendering.

This paper is organized as follows: in section II, we describe some works related to the development of augmented reality applications in iPhones; in Section III, we describe the edge detectors that were analyzed and considered in our investigation and point out its main properties and drawbacks; in section IV, we show how the considered edge detectors were adapted and optimized to the iPhone 3GS platform; results obtained with the adapted edge detectors implemented in the iPhone 3GS are presented in section V and, finally, in section VI, a conclusion and final considerations are presented.

II. RELATED WORK

Edge detection is a low-level image processing operation that is part of a large number of augmented reality applications. Surprisingly, image processing based approaches have not been the main choice when

one has to develop augmented reality applications on the Apple's iPhone platform. Since Apple only authorizes the implementation of augmented reality in the iPhone platform by using image overlays, most of its augmented reality applications use the integrated GPS technology, instead of image processing and pattern recognition methods. However, there are a few labs and research groups that are working with camera hacks to achieve augmented reality based on visual techniques.

The *Augmented Environments Lab* [3] [4] (AEL) has been working with augmented reality using QR patterns [10] recognition and AR Toolkit [1] to simulate a digital pet on the iPhone that responds to user interaction.

The *Simple-Image-Processing* [8] is an Objective-C library developed by Chris Greening, for an iPhone Sudoku application that photographs a Sudoku puzzle from a newspaper, and provides the correct answers, by processing the detected edges.

Similarly to [3] and [4], the main motivation of this work was to build a generic augmented reality application using image processing and computer vision methods. The software development requirements has led us to consider more carefully the basic image processing steps, in particular, the edge detection step. In the next sections, we review two of the most used edge detection methods, the Sobel and Canny edge detectors, and show how we have adapted and optimized them so that they could be used in mobile platforms.

III. EDGE DETECTION

Edge detection is one of the first steps in many pattern recognition and computer vision pipelines. It is an image processing technique that aims at the identification of sharp changes or discontinuities in the image brightness within neighbor regions on the image plane. The process itself can be done by using approximated derivatives usually implemented as digital filters.

Before performing edge detection some artifacts introduced in the input images by the capturing devices must be removed or at least reduced.

These artifacts include radial distortion, due to the characteristics of lenses of the cameras, illumination distortion, due to the ambient light frequency variation, shadows and noise introduced by the CCD or CMOS sensors.

In our project we considered only the radial distortion and noise problems due to sensors.

A. *Elliminating radial distortion*

Radial distortion is one of the main causes of problem in the camera calibration step. Hence, we considered investigating the effects of such problem considering the augmented reality system we implemented.

Fortunately, an intensive set of experiments has shown that the iPhone 3GS introduces minor radial distortion and due to the scope of this project, and the need to reduce processing time, we decided that the use radial distortion correction methods was not necessary.

B. *Elliminating and attenuating image noise*

Since the device generates images with random noise which must be eliminated or attenuated, the use of noise attenuating filters was required.

In this work, we started using a Gaussian Blur filtering with a 5x5 mask, in order to obtain the smoothing of the grayscale image generated from the colored input image.

Due to the device's memory and processing limitations, this method had to be optimized, so the filter was implemented as a separable Gaussian Filtering. A more detailed discussion will be presented in the next sections.

C. *Steps of edge detection*

According to *Trucco and Verri* [7], there are three steps required to perform edge detection. These steps are: *noise smoothing*, *edge enhancement* and *edge localization*. Noise smoothing, also known as noise reduction, aims at suppressing noise as much as possible, without destroying the edges of the image. On the other hand, edge enhancement tries to produce output images that have large intensity values at edge pixels and low intensity values elsewhere, so the edges can be easily located. The edge localization decides which of the local maxima in the filter's output are edges and which are just caused by noise.

D. *Sobel edge detector*

This method consists of a discrete differentiation operator, which computes an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical directions.

The inputs for the Sobel edge detector algorithm are formed by an image I and a threshold r. After the application of noise smoothing filters, the corresponding linear filter is applied to the new smoothed image Is, by using the masks shown in equation 1, producing two new images I_1 and I_2.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{(Eq 1)}$$

After this, the magnitude of the intensity gradient is estimated at each pixel $I(i,j)$, producing an image of gradient magnitudes as shown by equation 2

$$e_s(i,j) = \sqrt{I_1(i,j)^2 + I_2(i,j)^2} \quad \text{(Eq 2)}$$

Finally, all pixels $e_s(i, j)$ such that $e_s\ (i, j) > r$, are marked as edges.

E. *Canny edge extractor*

The Canny Edge Extraction Algorithm is an optimal edge detector having good localization property, such that the marked edges are as close as possible to the edges in the real image and minimal response, so that

one image edge should only be marked once, and only when there is strong evidence that it is really an important feature.

Given an image *I*, the first step on the algorithm is the *Canny Enhancer Algorithm*, which performs noise removal, gradient detection and edge strength analysis.

First of all, it is necessary to apply a Gaussian Smoothing to *I*, expressed as the convolution $J = I*g$ of *I* with a Gaussian kernel *g*.

After this, the following steps must be done for each pixel (i, j) in *J*:

a) Compute the gradient components J_x and J_y.

b) Estimate the edge strength e_s (equation 3).

$$e_s(i,j) = \sqrt{J_x(i,j)^2 + J_y(i,j)^2} \quad \text{(Eq 3)}$$

c) Estimate the orientation of the edge normal e_o (equation 4)

$$e_o(i,j) = a\tan\left(\frac{J_y}{J_x}\right) \quad \text{(Eq 4)}$$

The output is a strength image *Es,* formed by the values $e_s(i, j)$ and an orientation image *Eo,* formed by the values of $e_o(i, j)$.

After that, the *Non Maxima Suppression Algorithm* is applied to the strength image, considering the four directions identified by 0°, 45°, 90° and 135°, in order to remove non maxima points in the *Es* image and produce a new image *In*.

For each pixel (i,j) in *Es* it is necessary to find the direction d_k, which better approximates the direction $e_o(i, j)$, which is the edge normal. If $e_s(i, j)$ is smaller than at least one of its two neighbors along d_k, then $In(i, j) = 0$ otherwise $In(i,j) = e_s(i,j)$. The output image $In(i,j)$ consists of points belonging to thinned edges (i.e. *Es* after suppressing non maxima points).

The image produced by the previous step still contains local maxima originated by noise. In order to get rid of such local maxima, the next and last step of the process consists in using the *Hysteresis Threshold Algorithm*, which uses a minimum and maximum thresholding values t_l and t_h.

First, the next unvisited edge pixel, $In(i, j)$, such that $In(i, j) > t_h$ is located. In the sequel, the algorithm starts from $In(i, j)$, following the connected chains of local maxima in both directions perpendicular to the edge normal, as long as $In(i,j) > t_l$.

In the next step, all visited points are marked and a list of the locations of all points in the connected contour found is saved. The output is a set of lists, each one describing the position of a connected contour in the image, as well as the strength and the orientation images, describing the properties of the edge points.

IV. EDGE DETECTORS ON IPHONE

In order to implement both algorithms in the iPhone some optimization strategies had to be considered. Below we describe each of the optimization strategies that we have used so that the algorithms presented acceptable performances.

A. The Sobel edge detector iPhone approach

The Sobel algorithm itself is very simple and not much could be done to optimize its performance. Nevertheless, the Gaussian smoothing step was critical and its optimization led to considerable increases in performance.

We changed the 5x5 Gaussian mask filtering to a separable Gaussian filtering. By doing this, the number of elementary operations per pixel was reduced from 25 to 6, and the processing time was reduced from 225 average milliseconds, for a 320x480 image, to 102 average milliseconds.

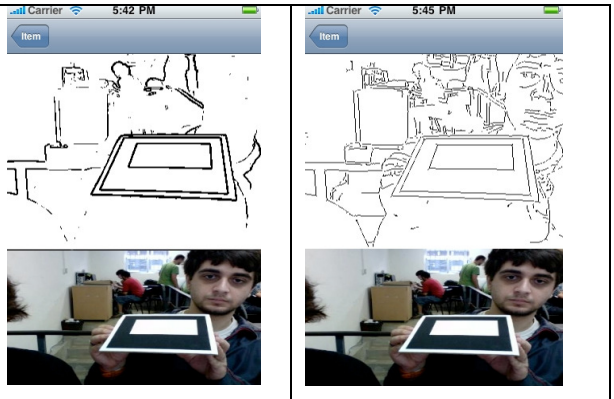
The Sobel algorithm combined with the separated Gaussian filter produced excellent results for the localization of the pattern features when using a threshold equal to 240 (Figure 3(a)). The main problem is that many edges were discarded what could be a problem for other applications.

B. The Canny edge extractor iPhone implementation

Due to the high cost of the Canny edge detector, the first step towards the optimization process was to apply a separable Gaussian filter [9], instead of a 5x5 mask to remove image noises, as we did with the detector based on the Sobel masks.

By doing this, the response time decreased, but not enough to be considered as a feasible solution to be used in an interactive application.

Figures 1(a) and 1(b) show a comparison of the results produced, respectively, by the Sobel and Canny Edge Extractor algorithm running on the iPhone. The images were resized after processing so they could fit the devices screen.



(a) Sobel edge detector on iPhone using a threshold value equal to 240 in a scale from 0 to 255.

(a)Canny edge Extractor on iPhone using 0.3 as minimum and 0.7 as maximum threshold value.

Figure 1 – Image filter results.

In order to improve the performance of the Canny Edge Extractor algorithm, its convolution matrices and the separable Gaussian filter were combined into one kernel. The results of this attempt yielded a decrease of approximately 75% on the response time for a 320x480 pixels image (Figure 2).

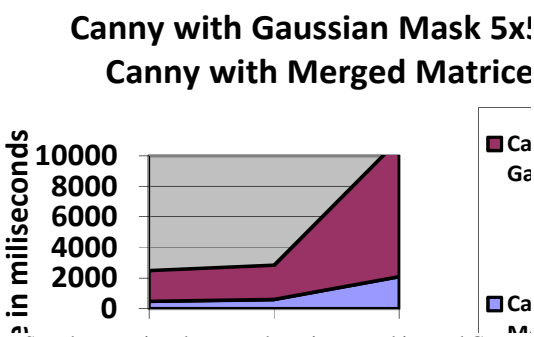


Figure 2 – Speed comparison between the noise smoothing and Canny edge extractor execution, versus the Canny edge extractor with merged matrices.

The results were excellent in terms of edge extraction. All edges were captured perfectly, nevertheless, there was still an excess of unwanted edges. The unwanted edges made difficult the calibration pattern recognition.

V. RESULTS

The charts in Figure 3 bellow show a comparison of the results in terms of the response times of the implemented solutions for the edge detectors.

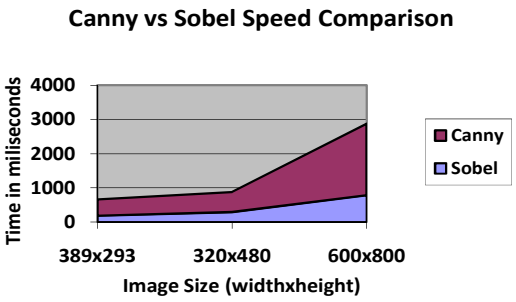


Figure 3 – Comparison of the time responses between the Canny edge extractor and the Sobel edge detector.

We tried to calibrate the lower and higher thresholds of the Canny edge detector, in order to eliminate undesirable edges that were hindering a perfect detection of the calibration pattern. Nevertheless, we could not achieve a robust and stable configuration.

After analyzing the results, we decided to base our edge detection solution, for the iPhone platform, on the Sobel method due the obtained performance and the qualitative results.

The overall results on the augmented reality project, using the Sobel edge detector can be seen on Figure 4 below.

Figure 4 – iPhone augmented reality.

VI. CONCLUSIONS

This work presented optimization strategies for edge detectors that enabled us to build an augmented reality application on the iPhone.

The lack of documentation for the iPhone imaging APIs was one of the biggest problems while trying to develop efficient filters and edge detectors.

Besides the many problems we have encountered during the project, we have achieved our purposes by developing an augmented reality application running on the iPhone that uses computationally expensive computer vision methods such as edge detection and camera calibration.

The experiments showed that edge detection was the major bottleneck of the application and the optimization strategies were fundamental for the success of the implementation.

Although the Canny Extractor was discarded for the augmented reality project its use should be considered for other applications which require more edges than the set produced by the Sobel detector implementation.

VII. ACKNOWLEDGMENTS

The authors would like to thank Erick Passos, Marcos Ramos, Eduardo Régis and Fernando Ribeiro. Anselmo Montenegro is grateful for the fund from FAPERJ under process number E-26/171.208/2006.

REFERENCES

[1] ARToolKit, http://www.hitl.washington.edu/artoolkit.

[2] J. Canny, A Computationl Approach to Edge Detection, IEEE Transactions on pattern analysis and machine intelligence., vol. PAMI-8, no 6, (1986).

[3] Augmented Environments Lab, http://www.Augmented environments.org /lab/.

[4] Arf, an Augmented Reality Virtual Pet on iPhone, Augmented Enviroments Lab, http://www.augmentedenvironments.org/lab/2008/11/28/arf-iphone-peek/, (2008).

[5] Computer Vision and Augmented Reality, Gattass M., website, http://www.tecgraf.puc-rio.br/~mgattass/ra/ra.html (2009)

[6] Augmented and Cooperative Virtual Reality, M. Alain, nausabin, http://www.tecgraf.puc-rio.br/~mgattass/ra/trb03/MichelAlain/ (2003)

[7] E. Trucco & A. Verri, "Introductory Techniques for 3D Computer Vision", Prentice Hall, (1998).

[8] Simple Image Processing, http://code.google.com/p/simple-ihpone-image-processing/

[9] L. G. Shapiro & G. C. Stockman, "Computer Vision", page 137~150. Prentice Hall, (2001).

[10] QR Code, http://en.wikipedia.org/wiki/QR_Code.

[11] R.Y.Tsai, *An Efficient and Accurate Camera Callibration (1986)*